DELAYED NEUTRONS FROM THE NEUTRON IRRADIATION OF ^{235}U

A Thesis

by

AARON DAVID HEINRICH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2008

Major Subject: Nuclear Engineering

DELAYED NEUTRONS FROM THE NEUTRON IRRADIATION OF ^{235}U

A Thesis

by

AARON DAVID HEINRICH

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	W. D. Reece
Committee Members,	Pavel Tsvetkov
	Jean-Luc Guermond
Head of Department,	Raymond Juzaitis

May 2008

Major Subject: Nuclear Engineering

ABSTRACT

Delayed Neutrons from the Neutron Irradiation of ^{235}U. (May 2008)

Aaron David Heinrich, B.S., Thomas Edison State College

Chair of Advisory Committee: Dr. W. D. Reece

A series of experiments was performed with the Texas A&M University Nuclear Science Center Reactor (NSCR) to verify ^{235}U delayed neutron emission rates. A custom device was created to accurately measure a sample's pneumatic flight time and the Nuclear Science Center's (NSC's) pneumatic transfer system (PTS) was redesigned to reduce a sample's pneumatic flight time from over 1,600 milliseconds to less than 450 milliseconds. Four saturation irradiations were performed at reactor powers of 100 and 200 kW for 300 seconds and one burst irradiation was performed using a $1.61 pulse producing 19.11 MW-s of energy.

Experimental results agreed extremely well with those of Keepin. By comparing the first ten seconds of collected data, the first saturation irradiation deviated ~1.869% with a dead time of 2 microseconds, while the burst irradiation deviated ~0.303% with a dead time of 5 microseconds. Saturation irradiations one, three and four were normalized to the initial count rate of saturation irradiation two to determine the system reproducibility, and deviated ~0.449%, ~0.343% and ~0.389%, respectively.

DEDICATION

Colossians 1:16 For by him all things were created: things in heaven and on earth, visible and invisible, whether thrones or powers or rulers or authorities; all things were created by him and for him.

ACKNOWLEDGEMENTS

Dr. W. D. Reece, thank you for your unlimited supply of help, encouragement and patience. You broadened my eyes to the importance of education.

Dr. Pavel Tsvetkov, thank you for your persevering willingness to help me in NUEN 404 and NUEN 601. I know you grew tired of seeing me. I thoroughly enjoyed each and every moment of your classes.

Dr. Jean-Luc Guermond, thank you for your mathematical expertise in MATH 602. This was the most difficult math class I have ever taken.

Jim Remlinger, thank you for modeling true leadership. I highly value the time spent with you at the NSC.

John Hernandez, thank you for your mathematical expertise and friendship. When is our next game of Risk?

Jared Porter, Jerry Newhouse, Pamela Gondeck, Wes Cullum, Travis Trahan and Sarah Schwartz, thank you for your support throughout the never-ending experiment.

Alfred Hanna, thank you for being my 'go-to' guy. Thank you for being so technology-savvy. Thank you for your hard work during the all-nighters. The experiment would have failed without you.

Tom Fisher, thank you for your electrical/electronic expertise and eccentric sense of humor.

Joe Snook and Jim Reynolds, thank you for your excellent mechanical know-how. Your machining and welding abilities are amazing.

Lindsay Welch, thank you for your friendship during your stay in College Station. I miss you.

David and Brenda Heinrich, thank you for your physical, emotional, spiritual and financial support throughout these past few years. You were the backbone of a successful stint at Texas A&M University.

Emily Heinrich, my beautiful bride. Thank you for choosing me to be your husband. Thank you for your continuing support throughout my numerous all-nighters at the NSC with Alfred repairing the reactor core sensor.

Lord Jesus, I thank You for Your faithfulness despite my shortcomings while I served in the military and studied at Texas A&M University.

NOMENCLATURE

DNP	Delayed Neutron Precursor
LANL	Los Alamos National Laboratory
LLNL	Lawrence Livermore National Laboratory
MCA	Multi-Channel Analyzer
NSC	Nuclear Science Center
NSCR	Nuclear Science Center Reactor
PTS	Pneumatic Transfer System
SCA	Single Channel Analyzer
TEU	Twenty-foot Containers
URL	Upper Research Level

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Page

Page

LIST OF TABLES

Page

I. INTRODUCTION

Following the terrorist attacks of the World Trade Center and Pentagon on September 11, 2001, the United States of America declared war on global terrorism. Though years have elapsed since the attack, the war on terror continues with numerous offensive conventional methods of counterterrorism, including the American and North Atlantic Treaty Organization troop militarization and invasion of countries aiding and abetting terrorist regimes, and the freezing and/or seizure of United States financial assets of "terrorist organizations, terrorist leaders, and corporations that serve as a front for terrorism and nonprofit organizations" [1]. Defensive conventional methods include the governmental imposition of stricter immigration laws, human tracking systems development, and heightened border security.

While counterterrorism is important, nuclear counterterrorism is paramount. The potential for illicit trafficking and subsequent detonation of nuclear weapons from terrorist organizations escalates as world economies continue to grow. For example, the U.S. Department of Transportation Bureau of Transportation Statistics states that the world maritime containerized traffic tripled in volume between 1995 and 2006 from 137 million to 417 million twenty-foot containers (TEUs) with an average annual growth rate of nearly 11%, as depicted by Table 1. The U.S. import traffic contribution from 2000 to 2005 almost doubled, from 6 million to 11.4 million containers, as shown in Table 2.

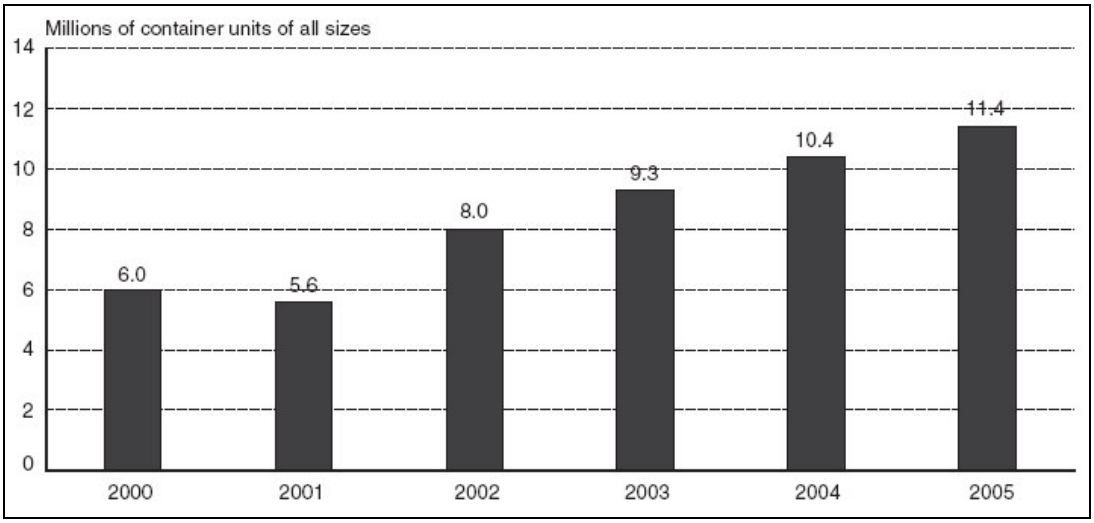
This thesis follows the style of *Physical Review C*.

Table 1. U.S. vs. World Maritime Container Traffic: 1995 – 2006. [2]

	World (millions)	United States (millions)	U.S. share of world total (percent)	U.S. rank
1995	137.2	22.3	16.3	1
1996	150.8	22.6	15.0	1
1997	160.7	24.5	15.3	1
1998	169.6	26.2	15.4	2
1999	184.6	28.0	15.2	2
2000	225.3	30.4	13.5	2
2001	236.7	30.7	13.0	2
2002	266.3	32.7	12.3	2
2003	305.0	36.3	11.9	2
2004	343.0	38.7	11.3	2
2005	378.0	42.0	11.1	2
2006[a]	417.0	46.3	11.1	2
Percent change, 1995-2006	203.9	107.2		
Average annual rate (percents), 1995-2006	10.6	6.8		

Increasing U.S. maritime commodity imports directly impact national security. To "prevent terrorists from accessing, using, or smuggling nuclear weapons – or the materials needed...to construct a nuclear weapon" [3], the Department of Energy, the Lawrence Livermore National Laboratory (LLNL) and the Weapons of Mass Destruction Directorate of the National Security Branch of the Federal Bureau of Investigation collaborated with interagency nuclear smuggling focus groups, U.S. Customs and Border Protection to protect U.S. seaports of entry.

Table 2. Maritime Containerized Imports into the United States: 2000 – 2005. [3]



A method proposed by LLNL utilizes a "neutron interrogation system" [4] that bombards the imported containerized cargo with neutrons. If the irradiated freight contains smuggled fissile material such as ^{235}U, the delayed neutron emission from the induced fissions will disclose the existence of illicit material.

A. Objective

To enhance nuclear weapons smuggling detection, the behavior of delayed neutrons, and particularly the shorter-lived ^{235}U delayed neutron groups, must be understood and verified; therefore, a series of experiments was performed with the Texas A&M University NSCR to verify these properties. Improved measurement of ^{235}U delayed neutron emission rates after thermally-induced fission was made possible by designing and creating an irradiation device that accurately measured a sample's pneumatic flight time, and redesigning the existing NSC PTS to significantly reduce its sample flight time to less than half a second.

B. Delayed Neutron Group Characterization History

Shortly after the discovery of fission in 1939, Roberts, Meyer and Wang [5] asserted that neutrons were emitted with a decay period of 12.5 ± 3 seconds up to a minute and a half after the neutron irradiation of uranium. Since the delayed emission could originate from either the direct disintegration or photodisintegration of the fission fragments, Roberts et al. [6] performed additional experiments and empirically determined the source to be the direct disintegration of the highly excited fission fragments. In the same year, Booth, Dunning and Slack [7] identified an additional group with a period of approximately 45 seconds, while Brostrom et al. [8] determined another between 0.1 – 0.3 seconds. Snell et al. [9,10] expanded the count to five in 1942, and within a couple of years de Hoffmann et al. [11] and Hughes et al. [12] identified a sixth.

In 1954, Los Alamos National Laboratory (LANL) initiated an extensive delayed neutron program that lasted over 20 years to comprehensibly define delayed neutron phenomena. In Keepin, Wimett and Zeigler's [13] experiment at LANL, computerized least-squares curve fitting of the ^{235}U delayed neutron emission found that a "six-group" model was sufficient to fit the experimental data. Their results are displayed in Table 3.

Table 3. 99.9% Enriched ^{235}U Thermal Fission Delayed Neutron Data.

Delayed Neutron Group	Relative Abundance (%)	Half-life (seconds)
1	3.3 ± 0.3	55.72 ± 1.28
2	21.9 ± 0.9	22.72 ± 0.71
3	19.6 ± 2.2	6.22 ± 0.23
4	39.5 ± 1.1	2.30 ± 0.09
5	11.5 ± 0.9	0.610 ± 0.083
6	4.2 ± 0.8	0.230 ± 0.025

C. Evaluated Nuclear Data Files

Since the 1950's additional studies have been performed to more accurately determine ^{235}U delayed neutron emission rates after thermally-induced fission. These experimental results have been published in a dedicated section of the Evaluated Nuclear Data Files [14] (ENDF). The ENDF contain fundamental information upon which neutron transport and nuclear phenomenon are calculated, and continue to be a work-in-progress because even the latest delayed neutron experimental data are still not in complete agreement.

The purpose of this experiment is to establish a fissile material irradiation system capable of performing reproducible irradiations and results, and to set error boundaries

on the experimental data itself, not any particular delayed neutron group parameter. A future research goal will be to verify the ENDF delayed neutron groups five and six data from other fissionable isotopes with the data collected at the NSC.

II. THEORY

A. Delayed Neutron Precursors

After ^{235}U absorbs a neutron, two significant mechanisms enable the excited nucleus to liberate its newly-found energy. The first mechanism is its de-excitation to the ground-state by gamma ray emission, and the second is fission that results in the immediate release of two to three neutrons on average. Fission fragments, neutrons, neutrinos, betas and an enormous amount of kinetic energy are also released.

While over 99% of these fission neutrons are "prompt" (emitted on the order of 10^{-13} seconds after fission), a small fraction are "delayed" neutrons that are released on the order of milliseconds to seconds after the subsequent decay of the daughter products generated from the beta-decay of the neutron-rich fission fragments. These fission fragments are known as delayed neutron precursors (DNP). Figure 1 displays an energy decay diagram of a typical delayed neutron precursor.

According to Pfeiffer, Kratz and Möller [16], at least 382 DNP isotopes have been identified. ^{91}Rb and ^{87}Br are the longest-lived known precursors with respective half-lives of 58.4 and 55.6 seconds, while ^{102}Rb and ^{101}Rb are the shortest-lived known with 37 and 32 millisecond half-lives, respectively. The continuously-growing number of identifiable isotopes indicates the apparent simplicity of the six-group delayed neutron model.

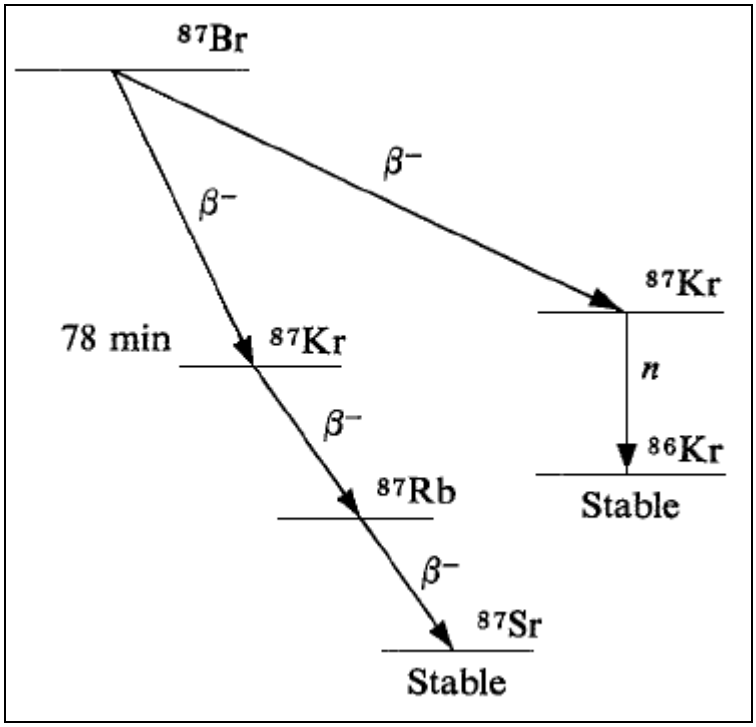


Figure 1. Energy Decay Diagram for ^{87}Br. [15]

B. Delayed Neutron Production

To estimate the delayed neutron properties of ^{235}U from first principles, a detailed knowledge of DNP formation is required. When binary fission occurs, the emitted fission fragments ordinarily possess different masses, resulting in a distribution of masses called the fission mass yield curve. For any given fissile material, the probability that an isotope of concern will be directly formed from fission can be seen in Fig. 2. Consequently, DNP production can either be produced directly from fission or indirectly from the decay of other fission fragment isotopes. The cumulative yield, or the summation of these direct and indirect probabilities, Y_c, can then be mathematically defined:

$$Y_c = \sum_k Y_i^k \tag{1}$$

where Y_i^k is the independent fission yield of isotope *k*.

By assuming the parent nuclides have negligible half-lives and the neutron absorption of the precursor is ignored, the time rate of change of the precursor *k* delayed neutron emission rate is as follows:

$$\frac{dN_k}{dt} = N^{U-235}\sigma_f^{U-235}\phi Y_c - \lambda_k N_k \tag{2}$$

where N_k is the number of precursor *k* atoms, $N^{U\text{-}235}$ is the number of ^{235}U atoms, $\sigma_f^{U\text{-}235}$ is the ^{235}U microscopic fission cross section, ϕ is the neutron flux and λ_k is the decay constant of precursor *k*. Equation 2 is a first order, linear, inhomogeneous, ordinary differential equation with constant coefficients and can be solved by the use of an integrating factor:

$$N_k(t) = \frac{N^{U\ 235}\sigma_f^{U\text{-}235}\varphi Y_c}{\lambda_k}\left[1 - \exp(-\lambda_k t)\right] \ . \tag{3}$$

If a sample is irradiated for time *T* and decayed for time t_d, the resultant precursor activity, A_k, is:

$$A_k(T, t_d) = N^{U-235}\sigma_f^{U-235}\phi Y_c \left[1 - \exp(-\lambda_k T)\right]\exp(-\lambda_k t_d) \ . \tag{4}$$

Assuming the delayed neutron emission rate is equal to the delayed neutron precursor decay rate, the six-group method of estimating delayed neutron emission for saturation irradiations is:

$$DN(T, t_d) = \sum_{k=1}^{6} N^{U\text{-}235}\sigma_f^{U\text{-}235}\phi Y_k \left[1 - \exp(-\lambda_k T)\right]\exp(-\lambda_k t_d) \ , \tag{5}$$

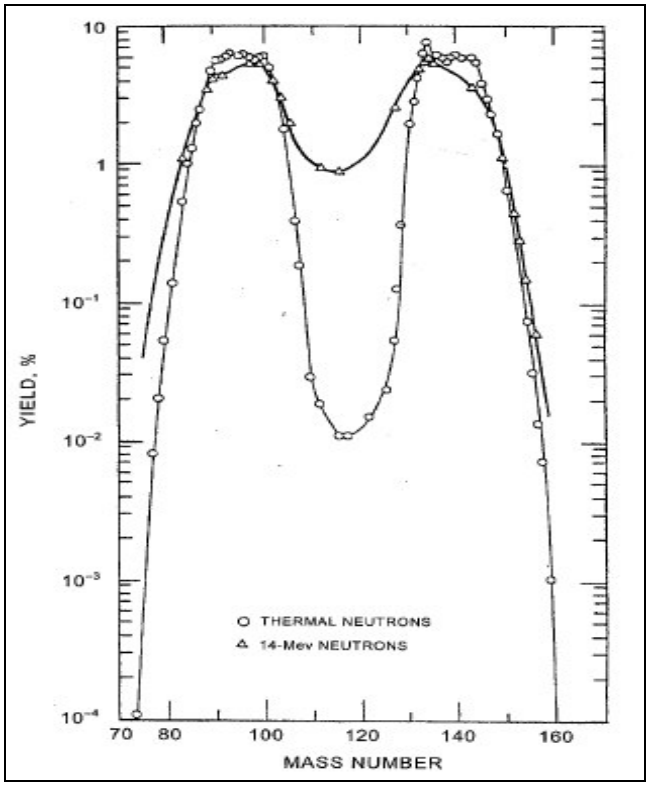


Figure 2. Fission Yield versus Mass Number for ^{235}U. [17]

where $DN(T,\ t_d)$ is the sum of the delayed neutron group emission rates and Y_k is the probability that a delayed neutron in group k will be produced from a fission event.

In burst irradiations, a Taylor series expansion is necessary for Eq. (5) because:

$$\lim_{T \to 0} DN(T, t_d) = 0 \quad , \tag{6}$$

After a first-order Maclaurin series expansion is performed on Eq. (5):

$$DN(t_d) = \sum_{k=1}^{6} N^{U\text{-}235} \sigma_f^{U\text{-}235} \phi Y_k \lambda_k \exp(-\lambda_k t_d) \quad . \tag{7}$$

Since the delayed neutron emission rate can be characterized by a sum of exponentials for both saturation and burst irradiations, the group abundances and decay constants can be determined by nonlinear, least-squares curve fitting when Eqs. (5) and (7) are plotted as functions of decay time.

C. Counting System Dead Time

In most radiation detection systems, a fraction of radiation interacting in the detection equipment will go unnoticed. A finite amount of time must exist between these interactions in the detection equipment for its associated electronics to discriminate between the events. This small amount of time is known as the counting system dead time. If high count rates exist in the detection system, such as delayed neutron detection immediately proceeding fissile material activation, corrections must be made to the measured data to account for these dead time losses.

Assuming the radiation detection system is non-paralyzable, or has fixed-length dead times, the true interaction rate can be calculated by knowledge of the measured interaction rate and the system dead time. Use of equation (8):

$$n - m = nm\tau \tag{8}$$

where n is the true interaction rate, m is the measured count rate and τ is the system dead time, and solving for n will determine the true interaction rate in the system:

$$n = \frac{m}{1 - m\tau} \quad . \tag{9}$$

III. EXPERIMENTAL APPARATUSES

Numerous components were required to perform the delayed neutron parameter measurements, including two ^{235}U samples, an array of three ^{3}He cylindrical neutron detectors, signal processing circuitry, the PTS, a reactor core sensor and a computerized control system.

A. Fissile Material

Isotope Products Laboratories produced the two ^{235}U samples used in the experiment. An 11.95 mg and a 12.27 mg sample of UO_2 were each homogeneously mixed into its own aluminum matrix pellet that has an active diameter and thickness measuring 4.88 mm and 1.02 mm, respectively. The front and back of each aluminum matrix pellet was encapsulated with a titanium cover 0.0508 mm thick. Both samples were enriched to 97.663 weight percent.

In case of a pellet rupture during its pneumatic transfer, each pellet was double encapsulated by two hermetically-sealed poly vials to prevent the escape of fission products. It is these poly vials, or 'rabbits', that are pneumatically inserted into and retrieved from the reactor core. Foam was placed above the top and below the bottom of the inner poly vial to absorb the impact of the rabbit with the pneumatic receivers. Figure 3 illustrates the rabbit construction; Fig. 4 displays two rabbits.

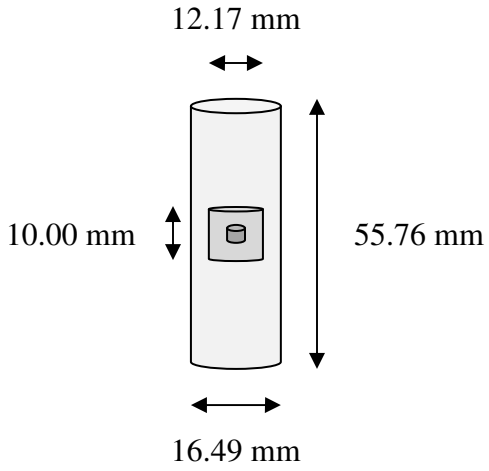


Figure 3. Double Encapsulated Hermetically-sealed Poly Vials.

Figure 4. Two Rabbits.

B. Delayed Neutron Detection System

Three LND Model 252 Cylindrical ^{3}He Neutron Detectors were used in the experiment to count the delayed neutrons. The ^{3}He detectors have an effective length and outer diameter of 28.27 and 2.54 centimeters, respectively, and an effective volume of 89.01 cubic centimeters. The detector casing and cathode materials are constructed with aluminum; each detector is pressurized to 3040 torr with a 99.91% ^{3}He isotopic concentration. Appendix A lists their physical specifications.

The detectors were installed in a 124 cm x 92 cm x 109 cm graphite block and cascaded so as to count a single sample through its entire decay, as shown in Fig. 5. Each was embedded in a 10.16 cm by 10.16 cm lead brick to suppress the gamma-ray pileup from the disintegration of the sample and its fission products. Figure 6 displays an overhead diagram with the physical dimensions.

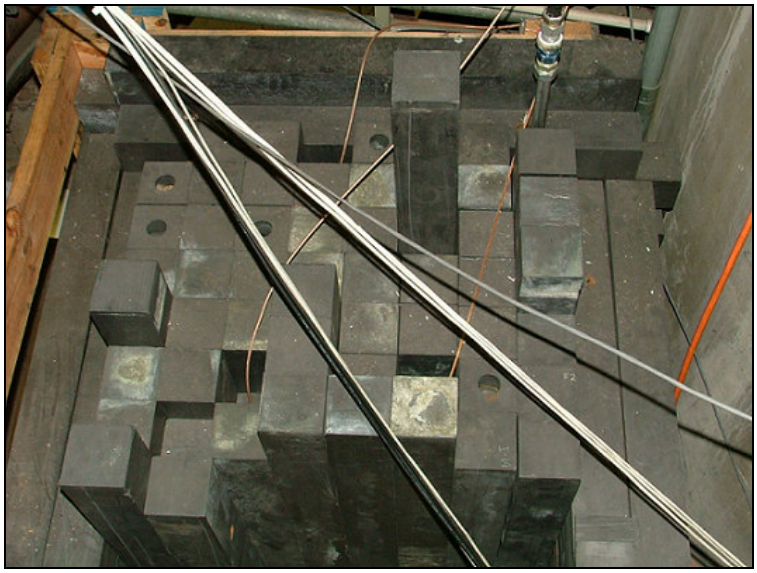


Figure 5. Graphite-moderated Counting Station.

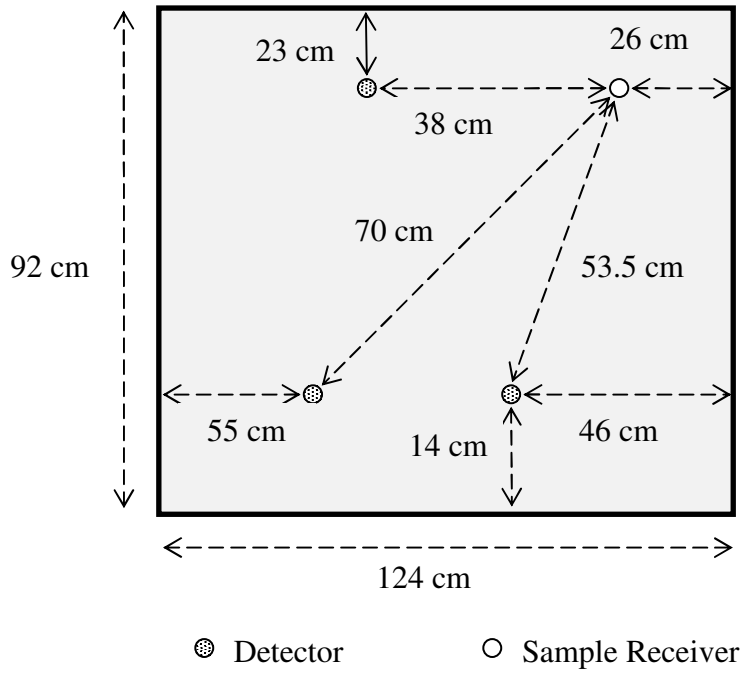


Figure 6. ^{3}He Detector Layout.

A block diagram of the electronic signal processing circuitry is depicted in Fig. 7. The detectors' output signals are first processed by their respective EG&G Ortec 142PC preamplifiers and then augmented by Tennelec TC 241 or Tennelec TC 242 amplifiers. Each output is then directed to its respective single channel analyzer (SCA), a Canberra SCA 2030 or Tennelec SCA TC450, then to Canberra Multiport II multi-channel analyzers (MCAs) where they are processed. Figure 8 shows the amplifiers installed in the Nuclear Instrumentation Module Bins.

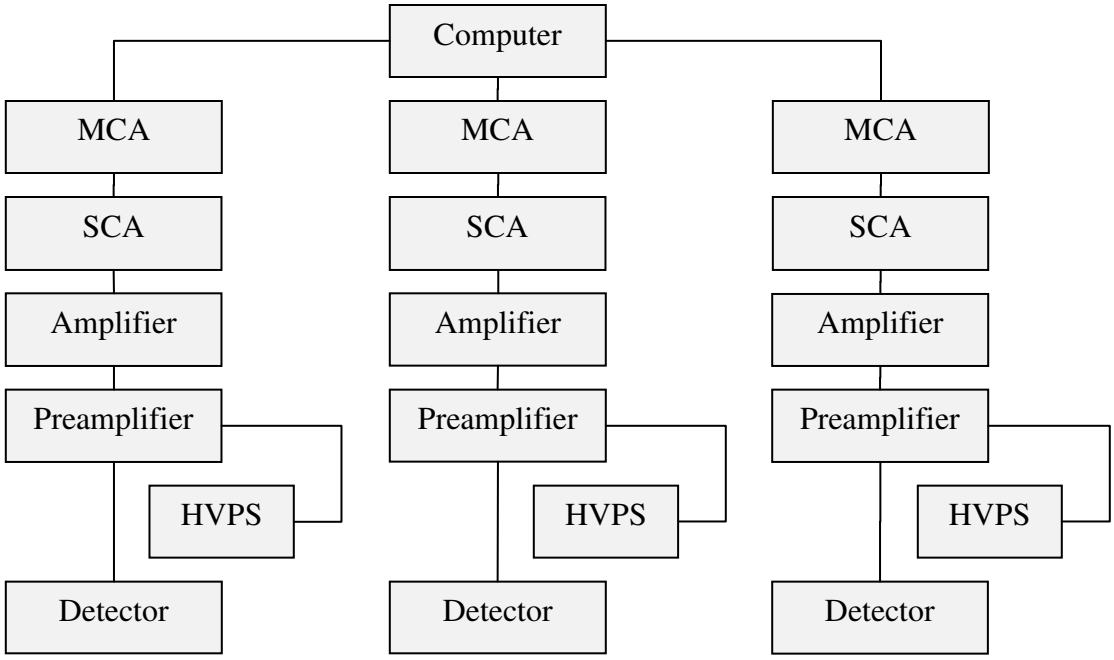


Figure 7. Block Diagram of the Signal Processing Circuitry.

Figure 8. Nuclear Instrumentation Module Bins.

The Canberra Multiport II MCA has 16,384 channels, each capable of measuring up to $2^{32} - 1$ counts. Its channel dwell times are user-adjustable between 1 microsecond and approximately 71 minutes.

MCA channel dwell times should be long enough to prevent significant Poisson variation in each channel's collected counts but short enough to ensure the collected counts exponentially decay over time. Also by considering that each irradiated sample was to be counted for only 350 – 400 seconds, the MCA channel dwell times were preset to 25 milliseconds per channel throughout the series of experiments.

C. Pneumatic Transfer System

The PTS, as depicted in Fig. 9, is composed of several key components, including two CO_2 supply systems, as seen in Fig. 10 and 11, two valve controllers, two pneumatic receivers and polyethylene tubing.

The primary firing station is located in the Upper Research Level (URL) of the NSC Confinement Building, the reactor core resides 26 feet below the water surface in the stall section of the pool, and the secondary firing station, counting station and storage box are situated in the Chase level. The storage box is displayed in Fig. 12. One pneumatic receiver is situated in the outer bayonet location of grid position A1 or A4 of the reactor core, as depicted in Fig. 13, and one in the counting station. Figure 14 displays a cross-sectional view of the NSC Confinement Building.

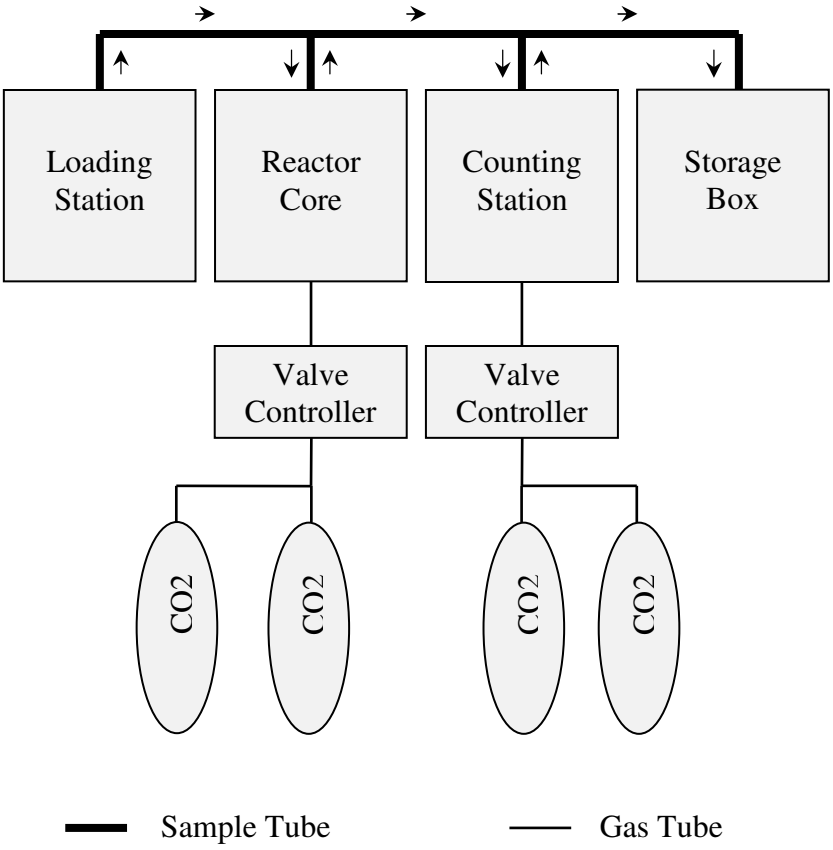


Figure 9. Pneumatic Transfer System.

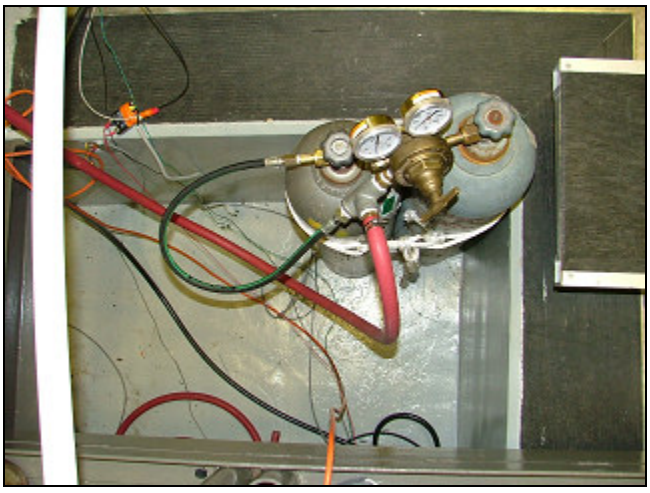


Figure 10. Primary Pneumatic Firing Station.

Figure 11. Secondary Pneumatic Firing Station.

Figure 12. Storage Box.

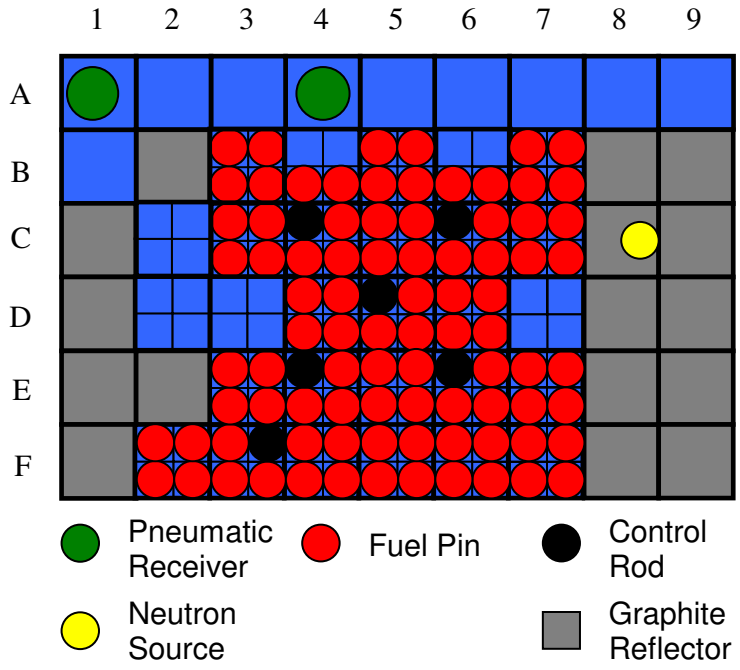


Figure 13. NSC Core VIII-A.

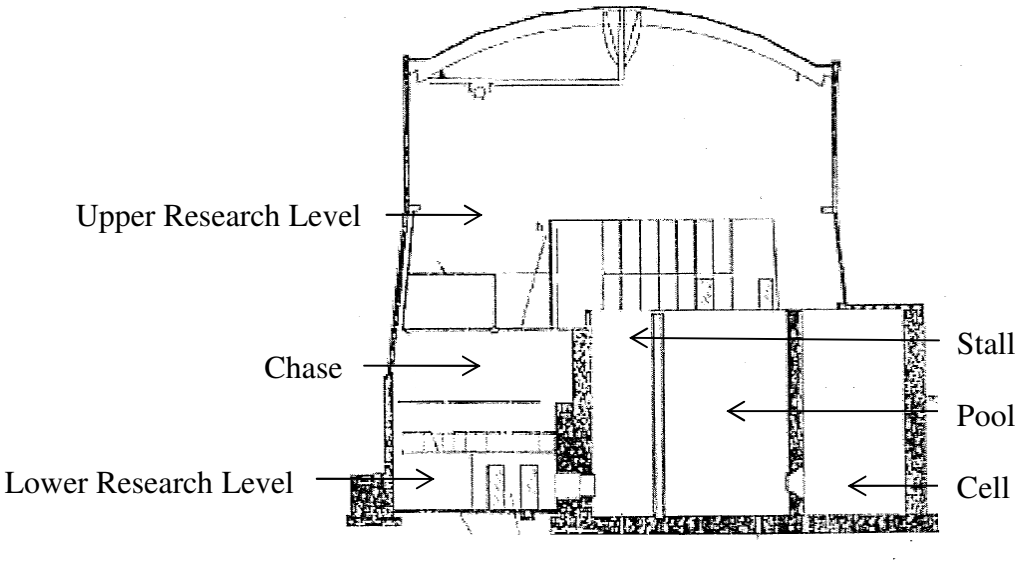


Figure 14. NSC Confinement Building.

CO_2 was chosen as the pneumatic gas because of its affordability and low neutron absorption characteristics, avoiding the radioactive isotopes that would be produced from compressed air. The two valve controllers operated solenoids to remotely control the flow of CO_2 through the tubing. By proper operation of the valves, the rabbit is transported from position to position in the PTS.

One of the goals in the PTS system setup was to reduce the rabbit flight time to less than half a second. This time is crucial to the calculation of the delayed neutron group characteristics due to the extremely short decay periods of groups five and six. Even with only 500 milliseconds of decay, group six activity will be ~ 22% of its end-of-irradiation value; its activity will be beyond accurate measure should more time elapse.

The rabbit flight time was decreased significantly by performing three actions. First, the distance the sample traveled from the reactor core to the counting station was reduced to approximately 44 feet. Second, a CO_2 supply station was added in the URL next to the reactor stall, thereby quartering the distance the CO_2 originally traveled to transfer the sample from the core to the counting station. Third, a surge tank was added in parallel to the new CO_2 firing station to supply a larger surge volume during pneumatic firing of the samples. All three of these actions resulted in reducing the average sample flight time from over 1,600 milliseconds to less than 450 milliseconds.

D. Reactor Core Sensor

In order to measure the shortest-lived delayed neutron emission rates, it is imperative to possess a flight time measurement system that accurately measures the elapsed time that transpires during a rabbit's flight from the reactor core to the counting

station. The flight time was accurately measured using a custom sensor placed in the reactor core and a photo-sensor in the counting station. Figure 15 displays the sensor.

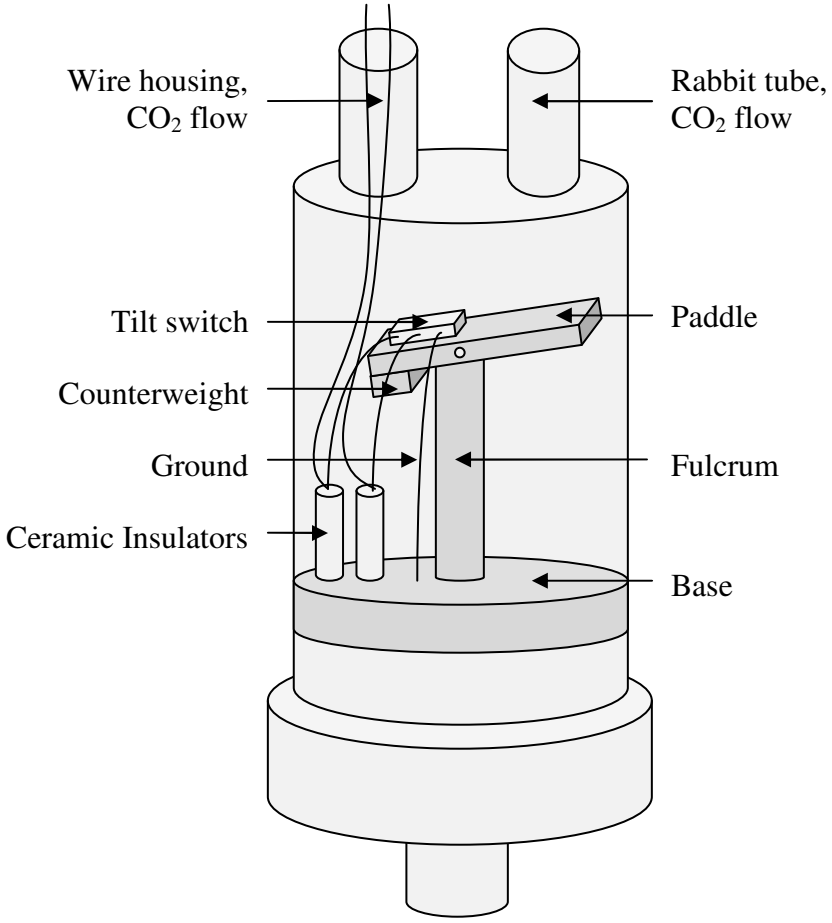


Figure 15. Reactor Core Sensor.

The custom sensor consists of 4 major parts: a ceramic tilt switch encased in watertight aluminum housing, a signal conditioning circuit, an amplifier circuit and a comparison circuit. The tilt switch, as seen in Fig. 16, is a ceramic sensor that operates similarly to a mercury switch but instead contains a non-metallic, alcohol-based,

conductive electrolyte. This switch manufactured by Spectron was chosen over a mercury switch because of its increased tilt resolution and lower neutron activation characteristics. Appendix B contains its technical specifications.

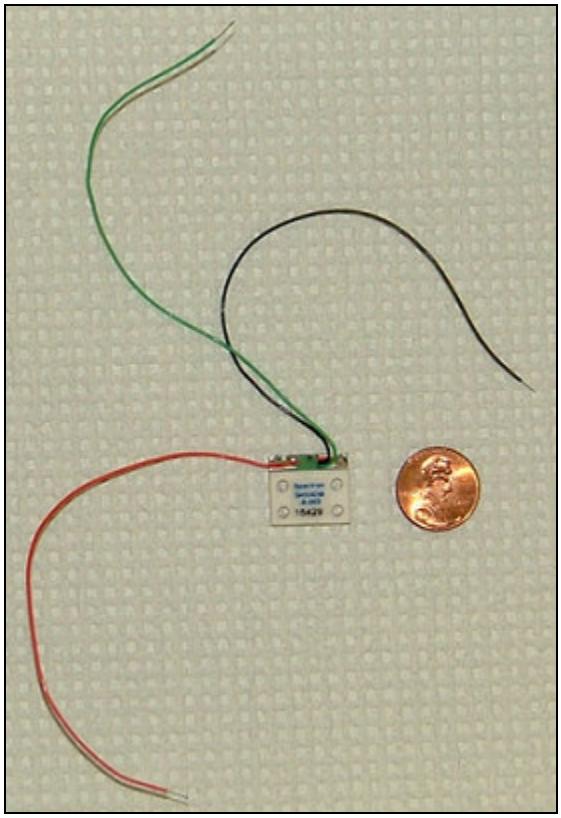


Figure 16. Ceramic Tilt Switch.

When the rabbit leaves the reactor core after a CO_2 burst, an electronic signal is created when the switch tilts. This signal is then converted from AC to DC, amplified into a larger signal and compared with a reference voltage in a comparison circuit. If the resultant signal is smaller than the pre-supplied reference voltage, the signal is sent to the

computerized control system to initiate the flight time timer. The timer is then secured when the rabbit disrupts the beam from the photo-sensor located in the counting station.

E. Computerized Control System

The experiment was administered by a C++ program written on a 3.00 GHz Dell Optiplex 170L computer with an Intel Pentium 4 processor. The solenoid operations, irradiation time and counting time were controlled by the customized program to automate the irradiation and counting sequence. Appendix C contains the source code.

The program initially prompts the user to identify the number of flight time sensors to be utilized in the irradiation sequence, as seen in Fig. 17; the program is also capable of execution solely for pneumatic testing purposes.

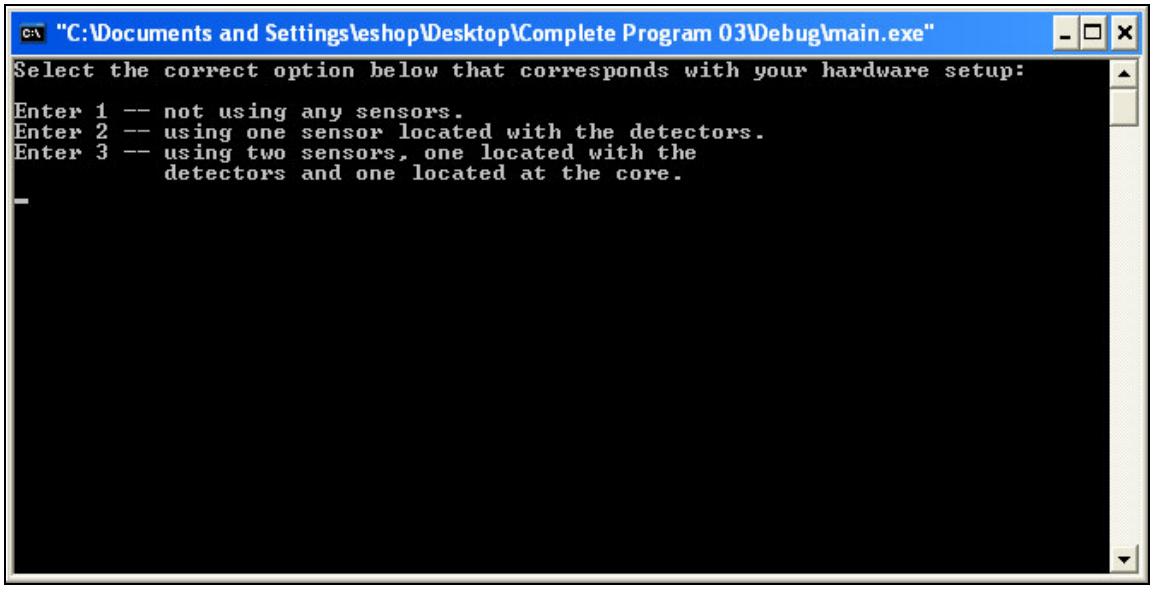


Figure 17. Number of Flight Time Sensors.

After a selection is made, the main menu is displayed to allow the user to execute the irradiation sequence, view the flight time results of an irradiation, test the solenoids or exit the program, as seen in Fig. 18.

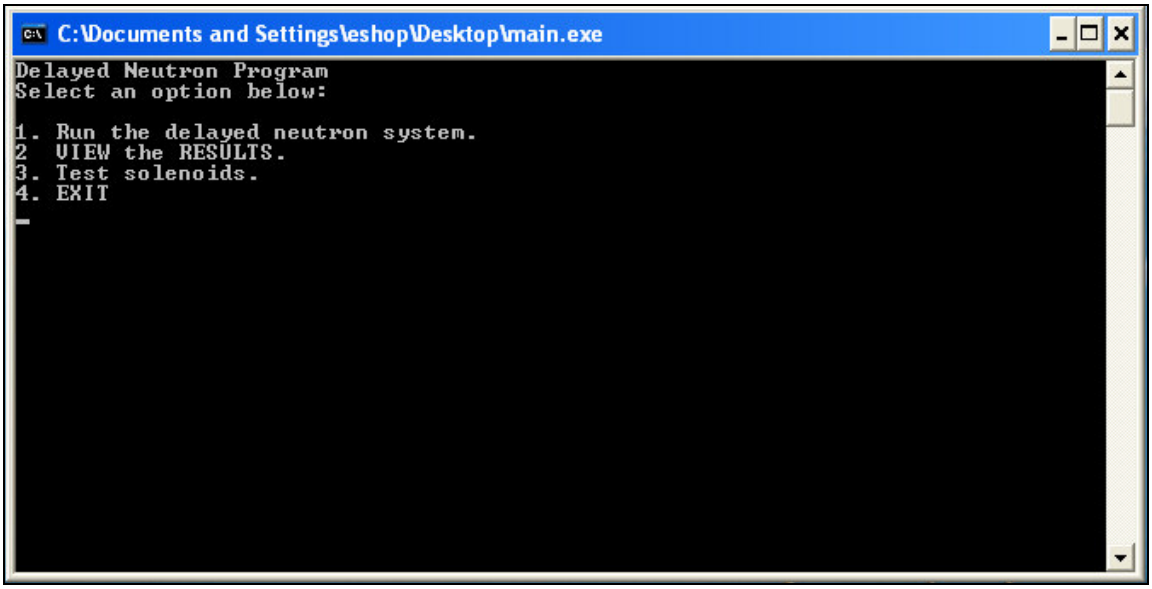


Figure 18. Main Menu.

If the solenoids are tested, the proper operation of each is verified and the main menu subsequently reappears. After the irradiation sequence option is chosen and the irradiation and counting times are entered in seconds, as shown in Fig. 19, the program permits the sample to be irradiated for the predetermined irradiation time. The sample is then pneumatically transferred to the counting station where the sample is counted, and then to the storage box for decay. Upon completion of the irradiation and counting sequence, the main menu is redisplayed to allow the user to view the flight time. Figure 20 displays the flight time measurement results.

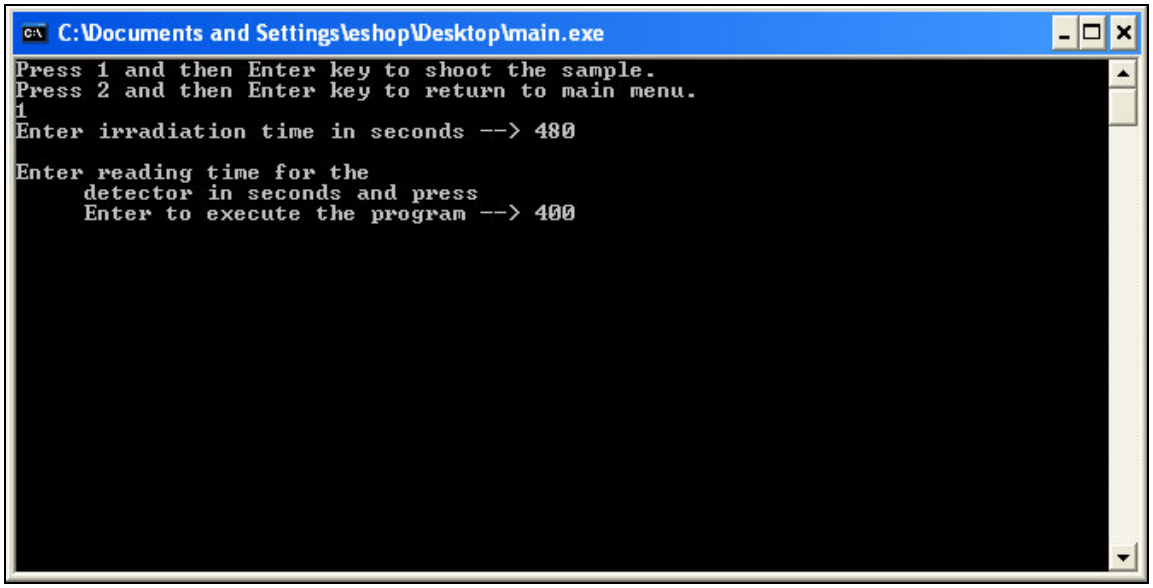


Figure 19. Irradiation and Counting Times.

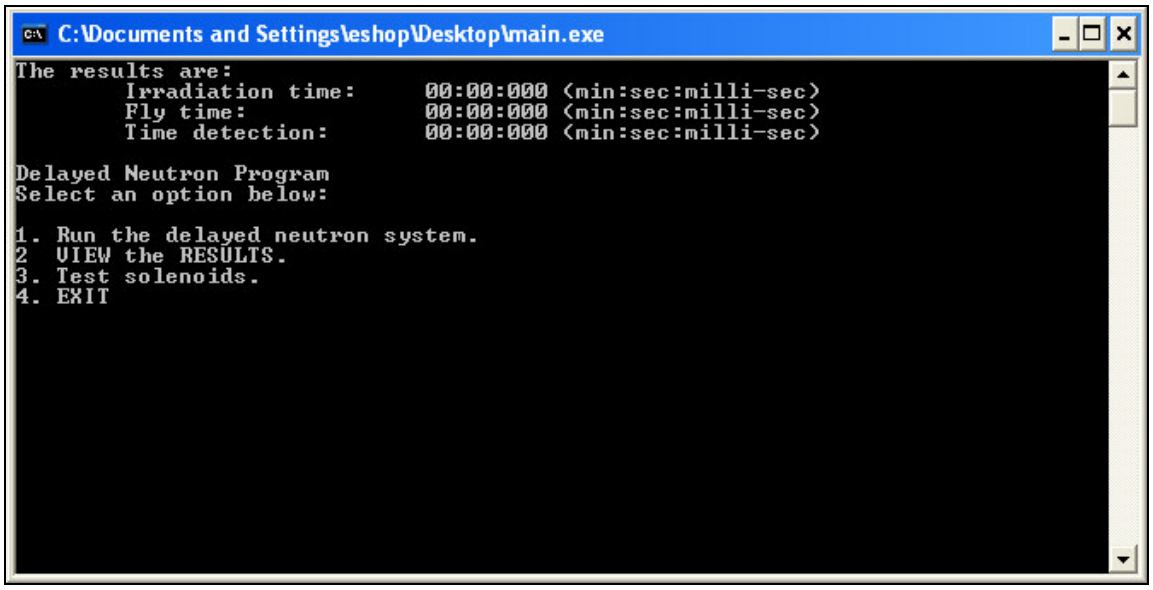


Figure 20. Flight Time Results.

IV. EXPERIMENTAL PROCEDURES

A. Saturation Irradiation

During the saturation irradiation phase, the desired irradiation and counting times were entered into the computerized control system. Since the period of the longest-lived delayed neutron group for ^{235}U is approximately 80 seconds, the samples were irradiated at a reactor power of 100 and 200 kW for 300 seconds to emphasize the contribution of the longer-lived delayed neutron groups by saturating the activity of all precursors.

After the rabbit was hand-loaded into the custom sensor, the tilt switch signal initiated the irradiation timer in the computerized control system. When the desired irradiation time had elapsed, the rabbit was pneumatically transferred from the reactor core to the counting station, where it was counted for 350 seconds to monitor all significant neutron emission. The rabbit was then transferred to the storage box for decay.

B. Burst Irradiation

In the burst irradiation phase, the irradiation process differs to accentuate the contribution of the shorter-lived delayed neutron groups. Ideally a sample should exit the core after the pulse is complete to maximize its neutron fluence but before the sample decays significantly. This window of opportunity was experimentally determined by graphing reactor power and the custom sensor tilt switch position as functions of time during simultaneous $1.61 reactor pulses and pneumatic retrievals. By superimposing these graphs, as seen in Fig. 21, a time delay of 110 milliseconds was

determined and inserted into the C++ program source code to delay the rabbit's pneumatic transfer.

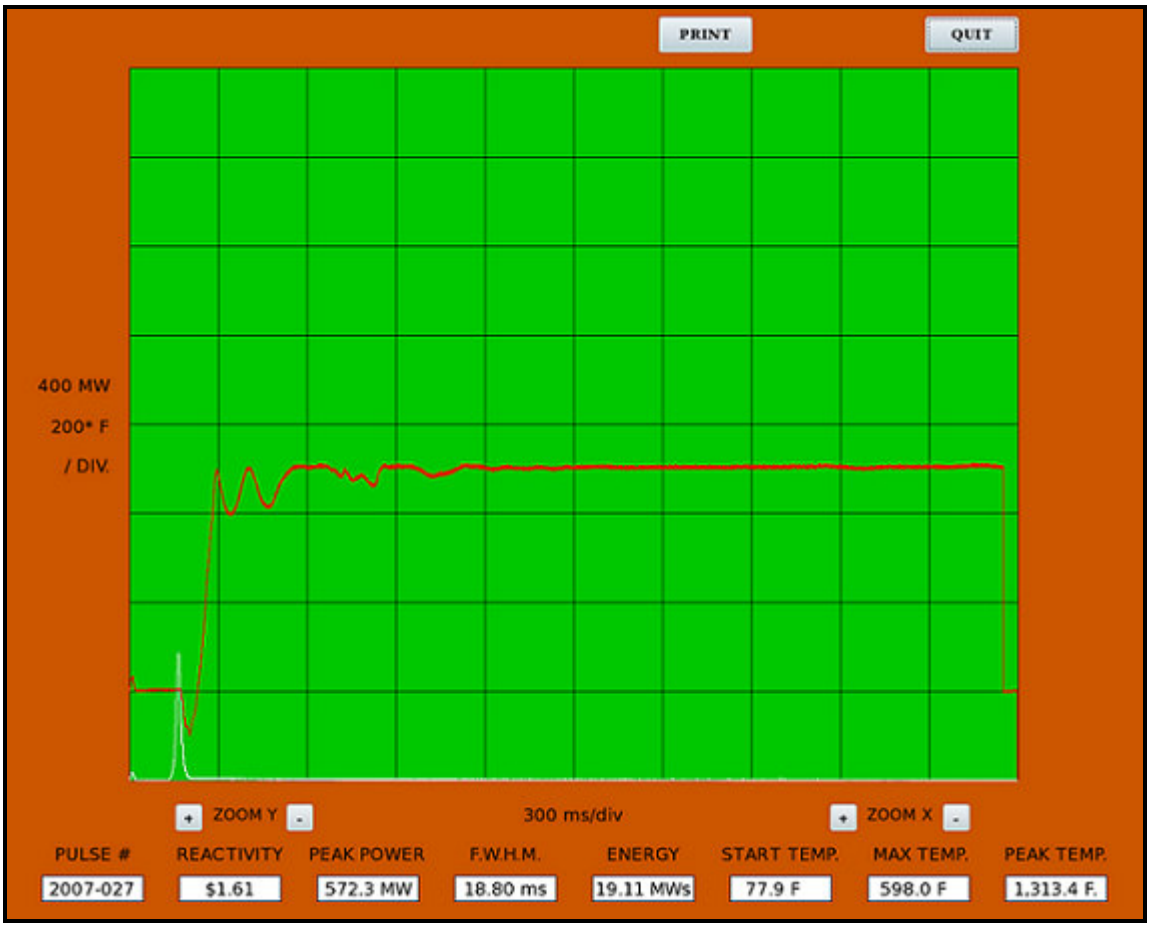


Figure 21. Reactor Power and Tilt Switch Position as Functions of Time.

The uppermost curve is the tilt switch position, while the lower is reactor power. The first spike in the reactor power curve is electronic noise created during the initiation of the reactor pulse, while the second is the reactor pulse itself. Solenoid operations to port compressed CO_2 to the NSCR transient control rod upon the reactor pulse initiation,

transient control rod transit time from full insertion in the core to full extraction and the controlled power excursion itself all contribute to the inherent delay of the reactor pulse.

When the reactor was operating at 300 watts the desired count time was manually inserted into the computerized control system and the rabbit was loaded as previously described. The reactor was then pulsed with $1.61 reactivity. After 110 milliseconds the irradiated rabbit was pneumatically transferred from the reactor core to the counting station where it was counted as previously described, then to the storage box for decay.

V. EXPERIMENTAL RESULTS

A. Saturation Irradiation Results

Four saturation irradiations were performed with the NSCR. Table 4 displays information for each irradiation. Figures 22 – 25 illustrate each MCA output during the irradiations. Each MCA channel had a dwell time of 25 milliseconds.

Table 4. Saturation Irradiations.

Pneumatic Insertion	Mass (mg)	Reactor Power (kW)	Irradiation Time (s)	Flight Time (s)	Count Time (s)
1	11.95	100	300	0.444	350
2	12.27	200	300	0.443	350
3	11.95	200	300	0.445	350
4	12.27	200	300	0.441	350

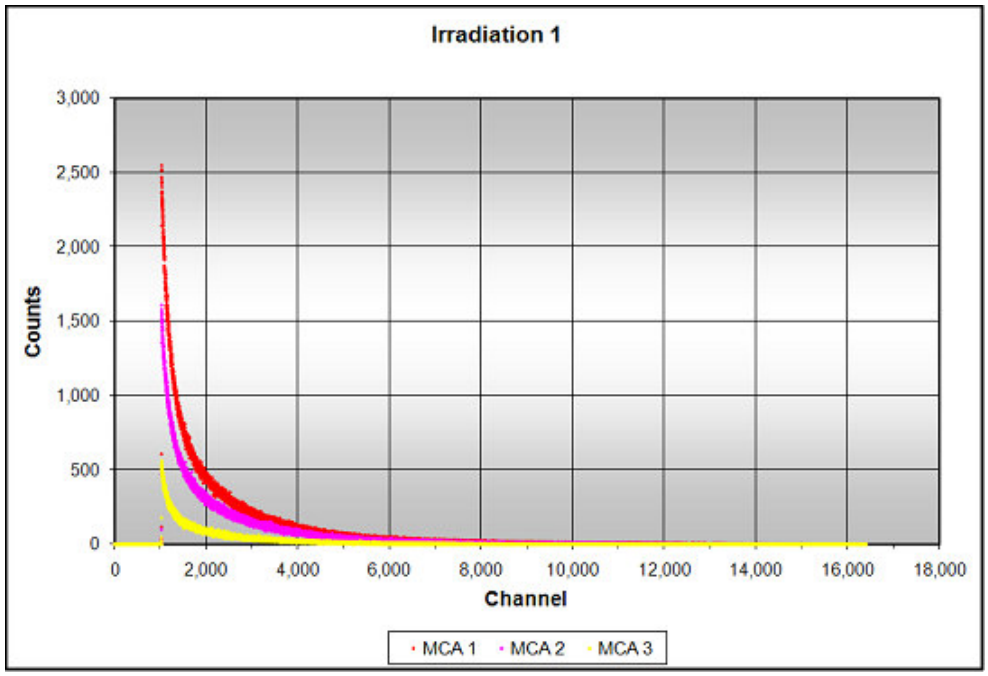


Figure 22. Saturation Irradiation 1.

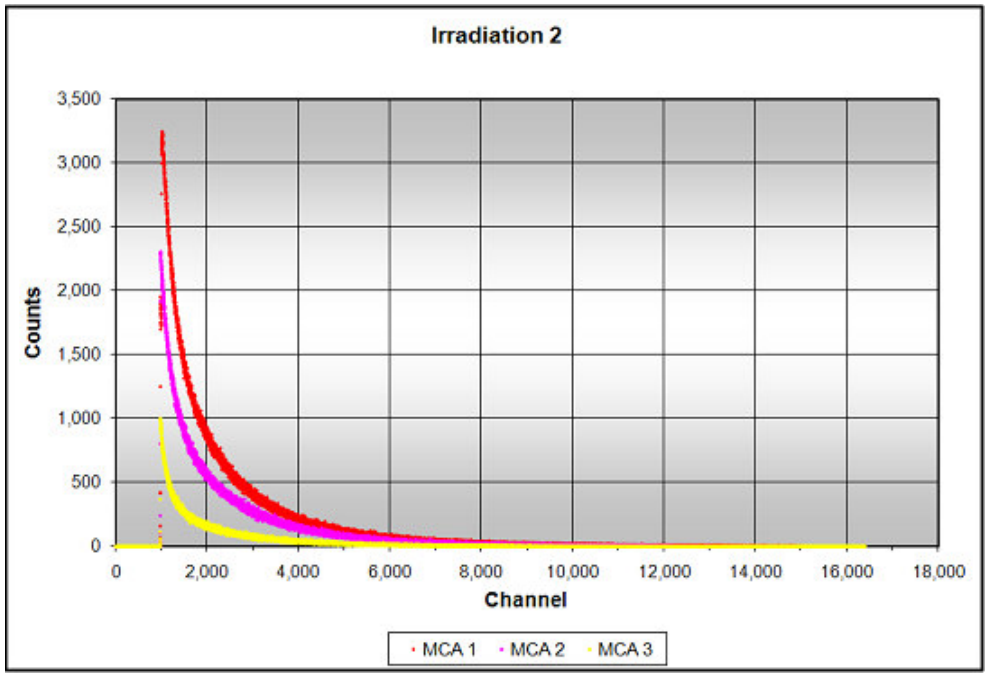


Figure 23. Saturation Irradiation 2.

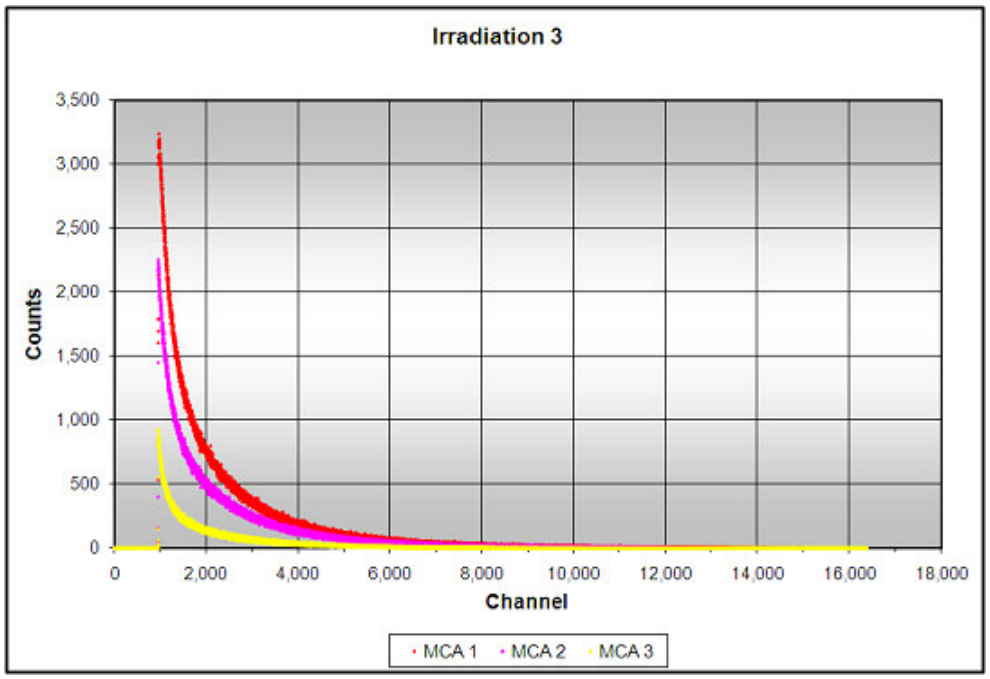


Figure 24. Saturation Irradiation 3.

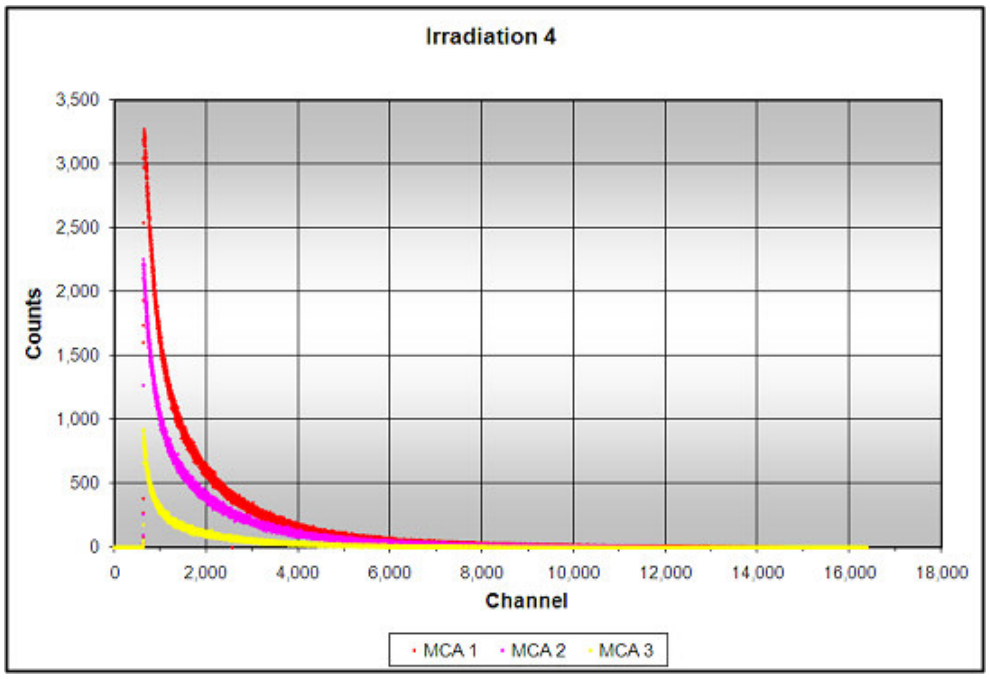


Figure 25. Saturation Irradiation 4.

B. Burst Irradiation Results

One burst irradiation was performed with the NSCR. Table 5 displays the burst information. Figure 26 illustrates each MCA output during the irradiation. Each MCA channel had a dwell time of 25 milliseconds.

Table 5. Burst Irradiation.

Pneumatic Insertion	Mass (mg)	Reactivity ($)	Full Width Half Max (ms^{-1})	Energy (MW-s)	Flight Time (s)	Count Time (s)
1	11.95	1.61	18.8	19.11	0.440	350

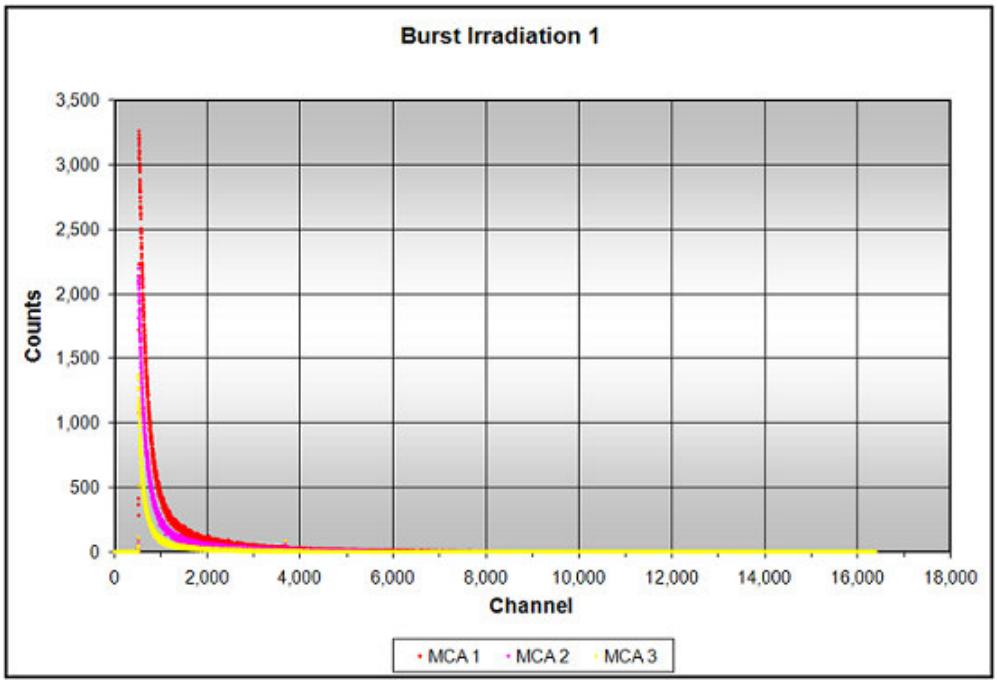


Figure 26. Burst Irradiation 1.

C. Optimally-weighted Count Ratios

The neutron detector distances were cascaded in the graphite counting station so as to count the emitted delayed neutrons throughout the entire decay of the irradiated samples. The three MCA outputs for each irradiation were combined by the following procedure to produce one decay rate curve.

The data during the first ten seconds of counting had 25 millisecond MCA channel dwell times; the data between 10 and 100 seconds and 100 seconds and greater were re-binned in half-second and five second bins, respectively, to minimize the inherent Poisson variation. Ratios of MCA 3 to MCA 1 output and MCA 3 to MCA 2 output were calculated and graphed as functions of time to determine when the two closest detectors were no longer overloaded by dead time, as evidenced by relatively constant MCA ratios. Figure 27 displays an example.

By using the universal error propagation formula:

$$\sigma_f^2 = \left(\frac{\partial f}{\partial x_1}\right)^2 \sigma_{x_1}^2 + \left(\frac{\partial f}{\partial x_2}\right)^2 \sigma_{x_2}^2 + \ldots + \left(\frac{\partial f}{\partial x_n}\right)^2 \sigma_{x_n}^2 \quad , \tag{10}$$

where f is a random function, x_1, x_2 and x_n are independent variables, errors propagated from the ratios:

$$\sigma_{MCA\ Ratio_{3/1,t}}^2 = \left(MCA\ Ratio_{3/1,t}\right)^2 \left(\frac{1}{MCA\ 3_t} + \frac{1}{MCA\ 1_t}\right) \tag{11}$$

and:

$$\sigma_{MCA\ Ratio_{2/1,t}}^2 = \left(MCA\ Ratio_{2/1,t}\right)^2 \left(\frac{1}{MCA\ 2_t} + \frac{1}{MCA\ 1_t}\right) \tag{12}$$

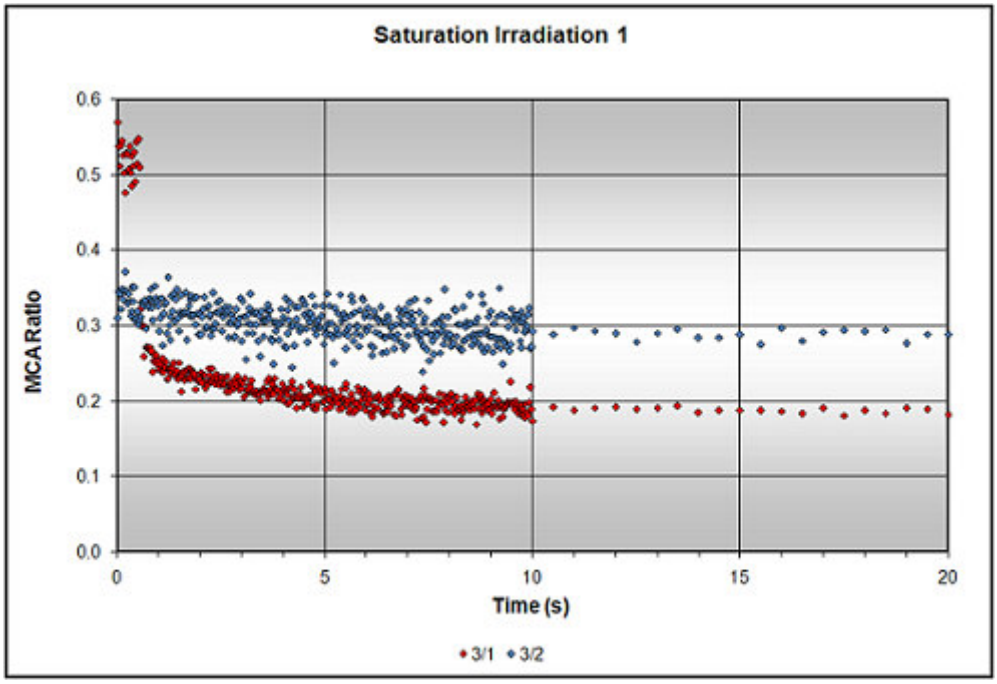


Figure 27. MCA Ratios as Functions of Time.

where σ^2 is the variance, were used to optimally-weight [18] the MCA ratios by giving more weight to the data with smaller error and less weight to the data with larger error. Equations (13) and (14) depict the process:

$$\left\langle MCA\ Ratio_{3/1} \right\rangle = \frac{\sum_{t=0}^{Irradiation\ Time} \left(\frac{MCA\ Ratio_{3/1,t}}{\sigma^2_{MCA\ Ratio_{3/1,t}}} \right)}{\sum_{t=0}^{Irradiation\ Time} \left(\frac{1}{\sigma^2_{MCA\ Ratio_{3/1,t}}} \right)} \tag{13}$$

$$\left\langle MCA\ Ratio_{2/1} \right\rangle = \frac{\sum_{t=0}^{Irradiation\ Time} \left(\frac{MCA\ Ratio_{2/1,t}}{\sigma^2_{MCA\ Ratio_{2/1,t}}} \right)}{\sum_{t=0}^{Irradiation\ Time} \left(\frac{1}{\sigma^2_{MCA\ Ratio_{2/1,t}}} \right)} \tag{14}$$

These optimally-weighted averages were used to scale the MCA 1 and MCA 2 output to the MCA 3 output after the time at which the two closest detectors were no longer overloaded by dead time. Table 6 displays the optimally-weighted averages for the saturation irradiations. All ratios were consistent.

Table 6. Optimally-weighted MCA Ratio Averages.

Irradiation Type	Irradiation Number	MCA 3 to 1 Ratio	Error (%)	MCA 2 to 1 Ratio	Error (%)
Saturation	1	0.19485	1.47E-05	0.28360	3.44E-05
Saturation	2	0.18812	7.79E-06	0.28723	1.99E-05
Saturation	3	0.18631	8.42E-06	0.27763	2.06E-05
Saturation	4	0.18007	7.94E-06	0.27402	2.03E-05

Only the MCA 3 output was utilized while detectors one and two were overloaded by dead time; the scaled outputs of MCA 1 and MCA 2 were averaged with the output of MCA 3 for the subsequent portion of the decay. When the resultant data were corrected for dead time and divided by their respective MCA channel dwell times, the delayed neutron count rates were determined as functions of time. Figures 28 – 32 display the decay rate curve for each irradiation.

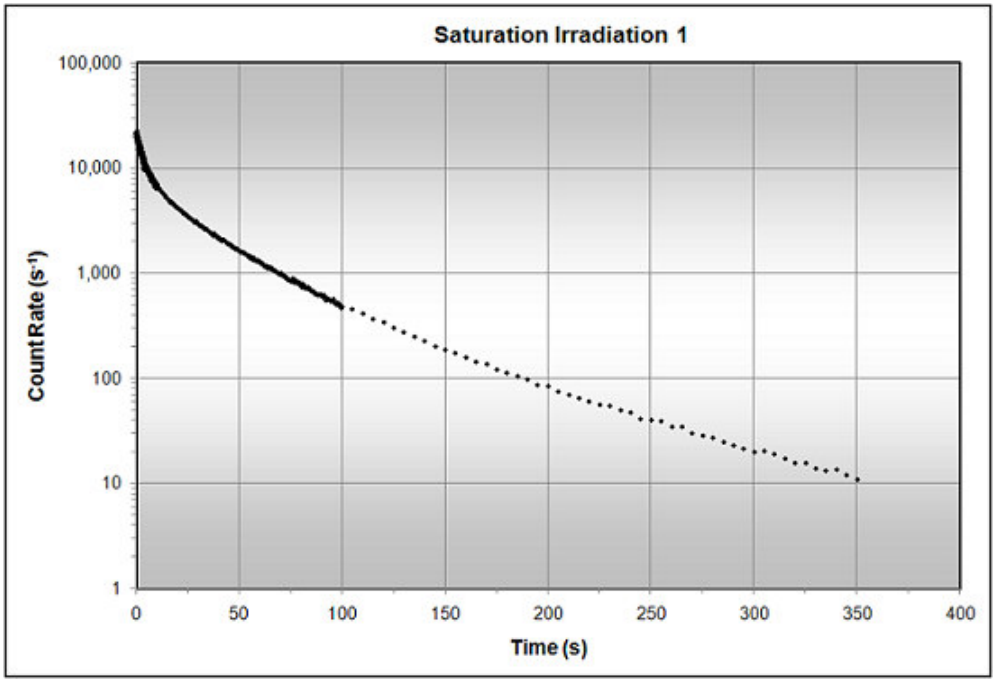


Figure 28. Combined Count Rate for Saturation Irradiation 1.

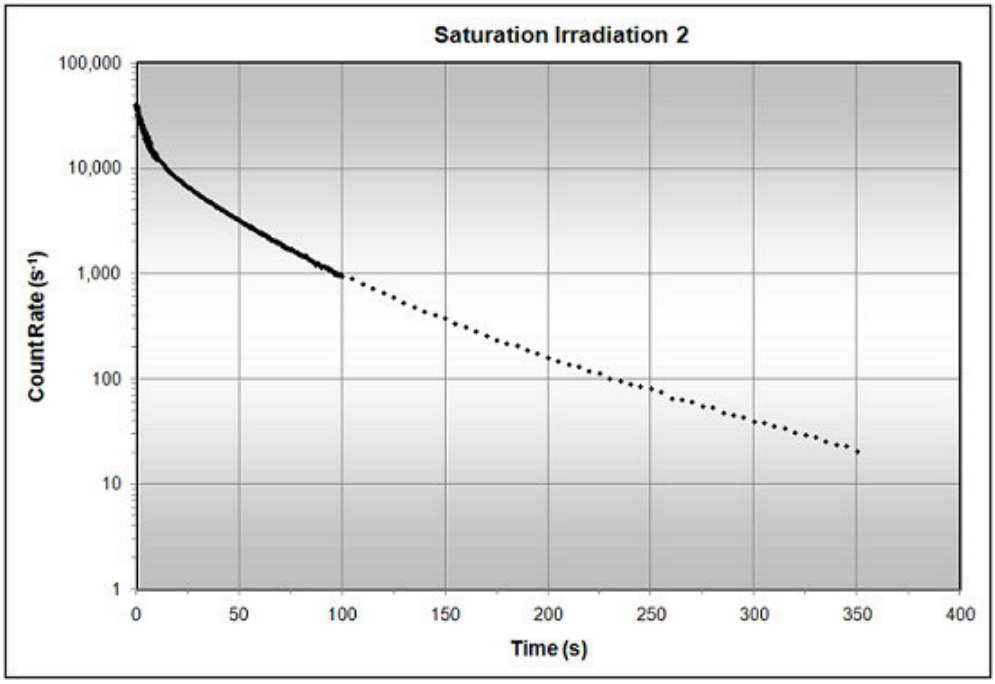


Figure 29. Combined Count Rate for Saturation Irradiation 2.

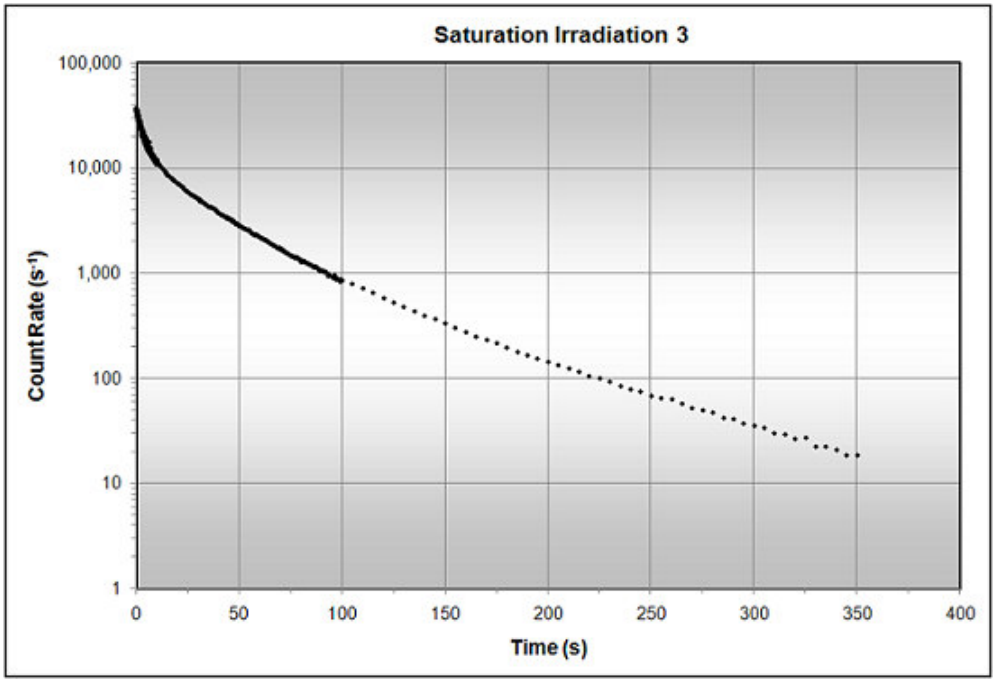


Figure 30. Combined Count Rate for Saturation Irradiation 3.

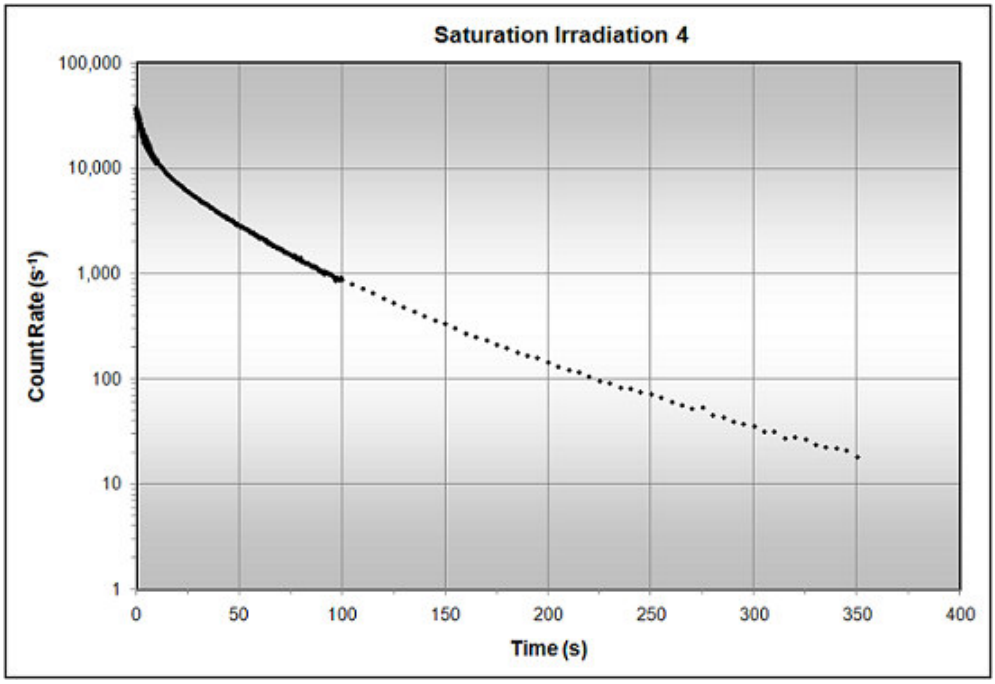


Figure 31. Combined Count Rate for Saturation Irradiation 4.

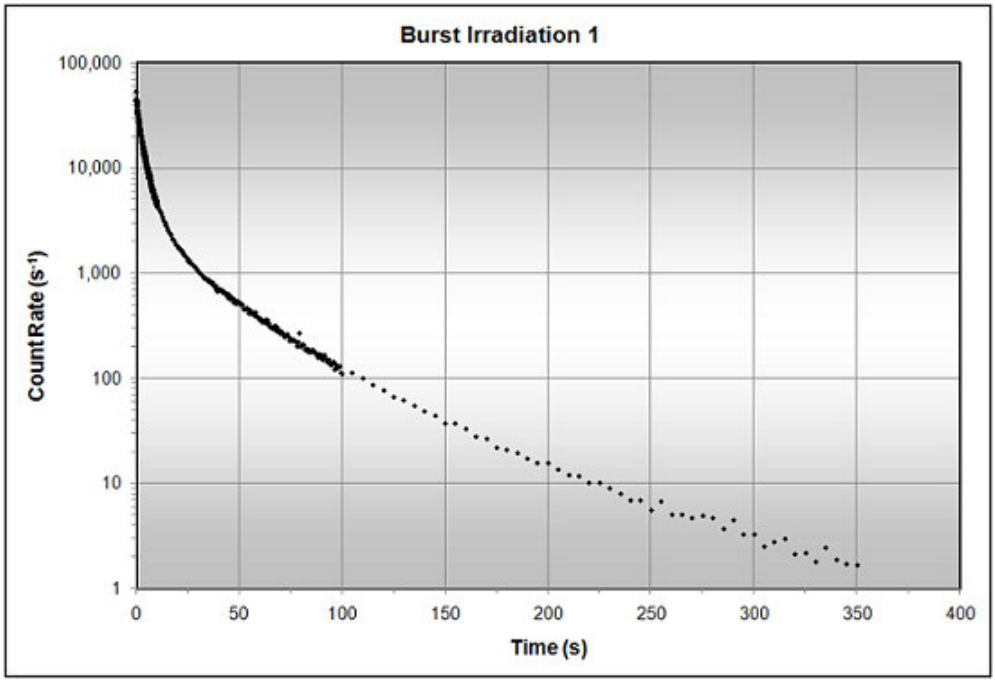


Figure 32. Combined Count Rate for Burst Irradiation 1.

D. Comparison to Keepin's Results

Keepin's results were reproduced by using the data in Table 3. By minimizing the sum of the squares of the differences between Keepin's and the dead time corrected experimental data, a statistical comparison was performed for both the saturation and burst irradiation results. Figures 33 and 34 display the comparisons.

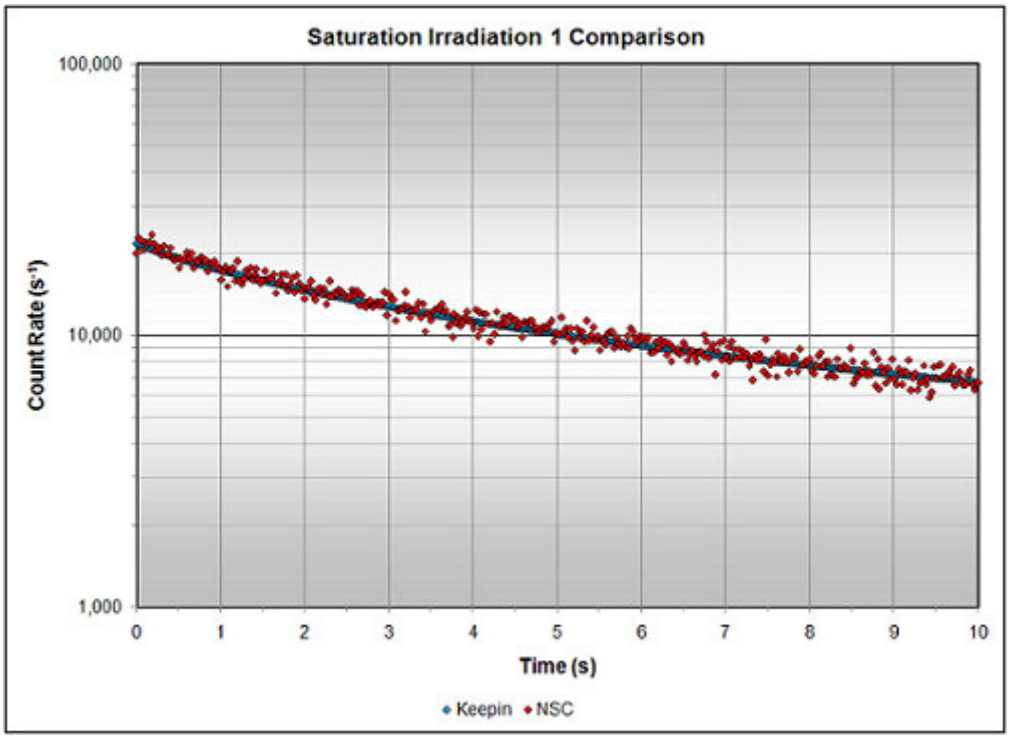


Figure 33. Saturation Irradiation 1 Comparison.

One statistical analysis that was performed on both sets of comparisons simply averaged the sum of the residuals during the first ten seconds of delayed neutron counting. The first saturation irradiation deviated ~1.869% with a dead time of 2 microseconds, while the burst irradiation deviated ~0.303% with a dead time of 5 microseconds. As can be seen, the experimental results agree extremely well with those of Keepin.

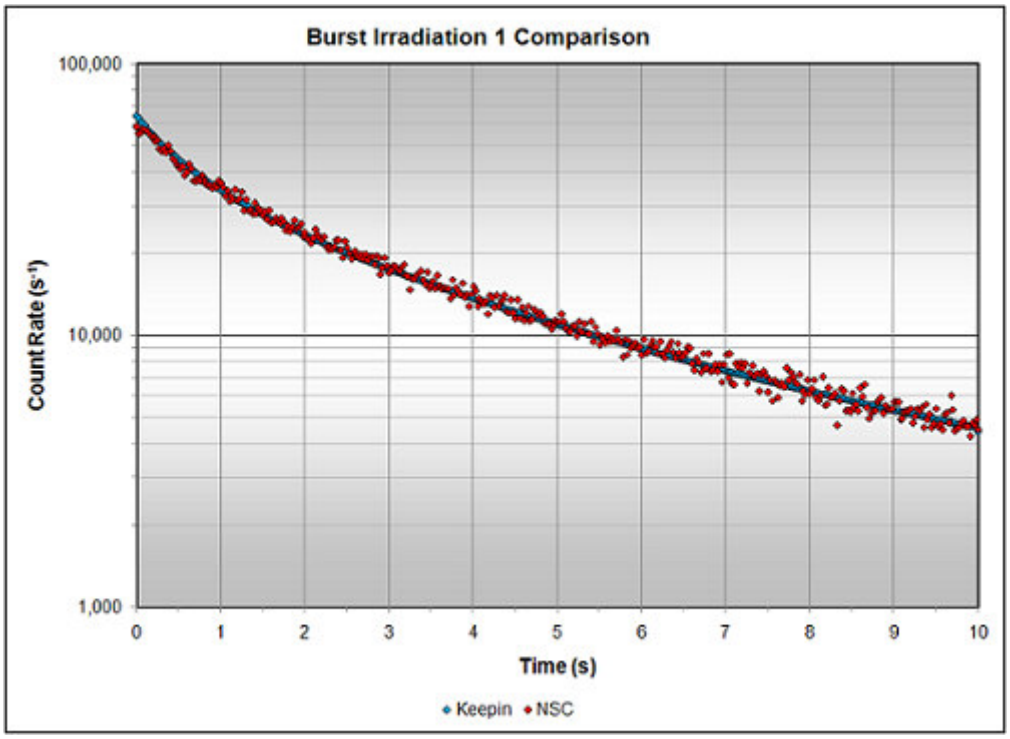


Figure 34. Burst Irradiation 1 Comparison.

E. Irradiation System Reproducibility

To gauge the consistency of the irradiation system, saturation irradiations one, three and four were normalized to the same initial count rate of saturation irradiation two by implementing the optimally-weighted average procedure as previously described. Figure 35 displays the normalized decay rate curves. By averaging the sum of the residuals, saturation irradiations one, three and four deviated from saturation irradiation two by 0.449%, 0.343% and 0.389%, respectively. These results confirm that the fissile material irradiation system is capable of performing consistently reproducible irradiation results.

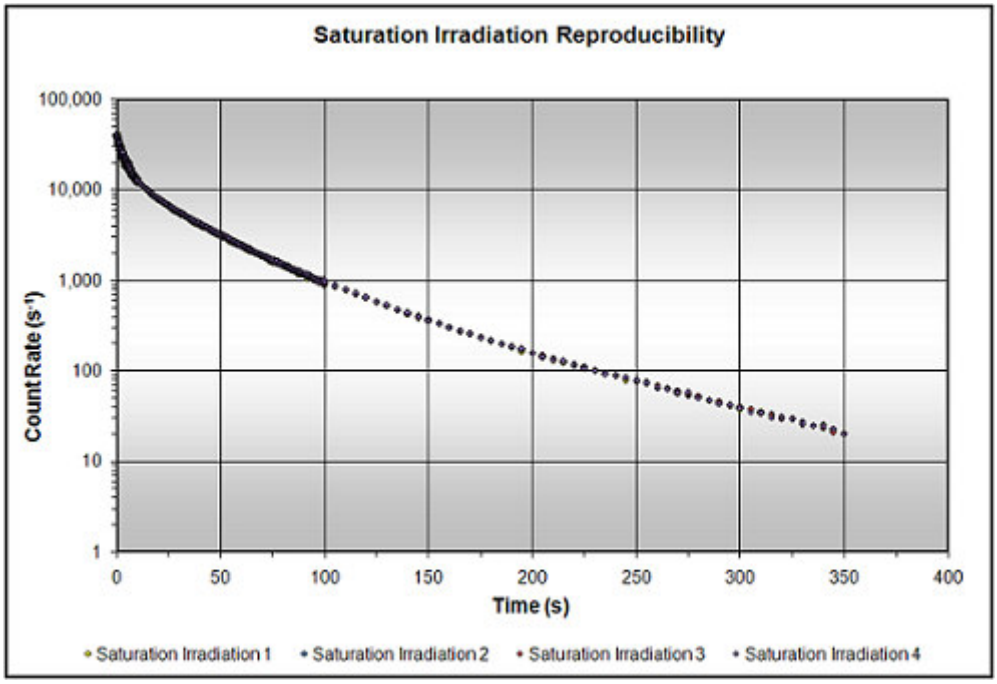


Figure 35. Saturation Irradiation Reproducibility.

VI. SUMMARY AND CONCLUSION

While world economies continue to grow a higher potential exists for nuclear weapons to be smuggled into U.S. seaports of entry, as indicated by the U.S. Department of Transportation Bureau of Transportation Statistics report. In order to inspect the maritime commodity importations for illicit nuclear material, LLNL suggested a method that irradiates the containerized cargo with neutrons. The existence of such fissionable isotopes such as ^{235}U will be declared by the delayed neutron emission from the induced fissions. By better understanding the behavior of the shorter-lived delayed neutron emission of ^{235}U, nuclear weapons smuggling detection will be enhanced.

At the Texas A&M University NSCR a series of experiments was performed to measure the behavior. Isotope Products Laboratories produced two uranium samples that were comprised of homogeneously-mixed UO_2 in aluminum matrix pellets. A delayed neutron detection system was assembled by utilizing three LND Model 252 cylindrical ^{3}He neutron detectors that were installed in a 124 cm x 92 cm x 109 cm graphite moderator and cascaded to enable the entire decay of an irradiated sample to be counted. To minimize the gamma-ray pileup each detector was embedded in a 10.16 cm by 10.16 cm lead brick. The electronic signal processing circuitry was comprised of numerous preamplifiers, amplifiers, SCAs and MCAs. The pre-existing NSC PTS was overhauled to reduce a sample's average flight time by a factor of four to less than 450 millseconds. A custom reactor core sensor was developed to allow the pneumatic flight time of a sample to be accurately measured. A custom C++ program was written to control the conduct of the saturation and burst irradiations.

The saturation irradiations emphasized the contribution of the longer-lived delayed neutron groups. The samples were irradiated for 300 seconds at reactor powers of 100 and 200 kW and counted for 350 seconds to monitor their entire decay. The shorter-lived delayed neutron groups were accentuated during the burst irradiation. The sample was irradiated when $1.61 reactivity was promptly inserted into the reactor core and counted as previously described. The sample exited the reactor core 110 milliseconds after the pulse was initiated to maximize the sample's neutron fluence before being counted.

The three MCA outputs for each irradiation were combined to produce one decay rate curve. MCA channel dwell times were adjusted to minimize the Poisson statistical uncertainty and MCA output ratios were calculated to determine when the two closest detectors were overloaded by dead time. Optimally-weighted ratio averages were then computed to properly weight the MCA output data. The output of MCA 1 and MCA 2 were scaled with the optimally-weighted ratio averages after the time their detectors were no longer affect by dead time; the three MCA outputs were then averaged. After correcting the resultant data points for dead time and dividing them by their respective MCA channel dwell times, the combined decay rates were determined as functions of time.

Simple statistical comparisons were performed for both the saturation and burst irradiation results. Experimental results agreed extremely well with those of Keepin. By averaging the sum of the residuals during the first ten seconds of delayed neutron counting, the first saturation irradiation deviated ~1.869% with a dead time of 2

microseconds, while the burst irradiation deviated ~0.303% with a dead time of 5 microseconds. Saturation irradiations one, three and four were also normalized to the initial count rate of saturation irradiation two to determine the reproducibility of the irradiation system, and deviated ~0.449%, ~0.343% and ~0.389%, respectively. These minimal deviations proved the irradiation system's stability.

REFERENCES

[1] G. Bush, "President Freezes Terrorists' Assets," The White House, September 24, 2001. <http://www.whitehouse.gov/news/releases/2001/09/20010924-4.html>.

[2] U. S. Department of Transportation, Research and Innovative Technology Administration, Bureau of Transportation Statistics, "America's Container Ports: Delivering the Goods," (March 2007).

[3] U. S. Federal Bureau of Investigation, Counterterrorism Division, "NUCLEAR TERRORISM: What we're doing to prevent it," (October 2005).

[4] D. Sprouse, "Screening Cargo Containers to Remove a Terrorist Threat," *Science and Technology Review*, (May 2004).

[5] R. Roberts, R. Meyer, L. Hafstad, and P. Wang, "Further Observations on the Splitting of Uranium and Thorium," *Physical Review*, **55**, 510 (1939).

[6] R. Roberts, R. Meyer, L. Hafstad and P. Wang, "The Delayed Neutron Emission which Accompanies Fission of Uranium and Thorium," *Physical Review*, **55**, 799 (1939).

[7] E. T. Booth, J. R. Dunning, and F. G. Slack, "Delayed Neutron Emission from Uranium," *Physical Review*, **55**, 876 (1939).

[8] K. J. Brostrom, J. Koch, and T. Lauritsen, "Delayed Neutron Emission Accompanying Uranium Fission," *Nature,* **144**, 830 (1939).

[9] A. H. Snell, V. A. Nedzel, H. W. Ibsen, J. S. Levinger, R. G. Wilkinson, and M. B. Sampson, "Studies of the Delayed Neutrons I: The Decay Curve and the Intensity of the Delayed Neutrons," *Physical Review*, **72**, 541 (1947).

[10] A. H. Snell, J. S. Levinger, E. P. Meiners, M. B. Sampson and R. G. Wilkinson, "Studies of the Delayed Neutrons II: Chemical Isolation of the 56-Second and the 23-Second Activities," *Physical Review*, **72**, 545 (1947).

[11] F. De Hoffmann, B. T. Feld, and P. R. Stein, "Delayed Neutrons from U^{235} after Short Irradiation," *Physical Review*, **74**, 1330 (1948).

[12] D. J. Hughes, J. Dabbs, A. Cahn, and D. Hall, "Delayed Neutrons from Fission of U^{235}," *Physical Review*, **73**, 111 (1948).

[13] G. R. Keepin, T. F. Wimett, and R. K. Zeigler, "Delayed Neutrons from Fissionable Isotopes of Uranium, Plutonium and Thorium," *Physical Review*, **107**, 1044 (1957).

[14] W. B. Wilson, and T. R. England, Delayed Neutron Study Using ENDF/B—VI Basic Nuclear Data, Los Alamos National Laboratory, Los Alamos, NM, 2000.

[15] W. H. Stacey, *Nuclear Reactor Analysis*, 1st Edition, (John Wiley & Sons, New York, 2001).

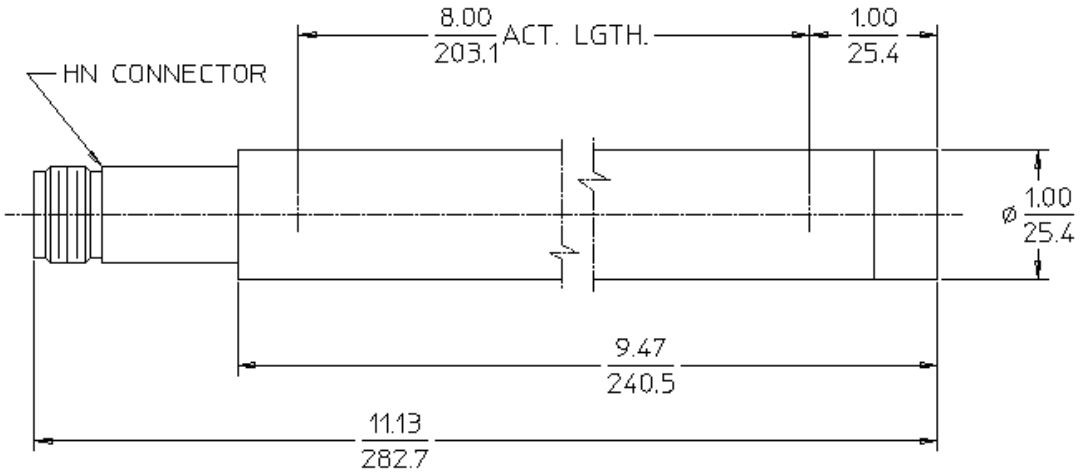
[16] B. Pfeiffer, K. Kratz, and P. Möller, Status of Delayed-Neutron Precursor Data: Half-Lives and Neutron Emission Probabilities, Los Alamos National Laboratory, Los Alamos, NM, 2001.

[17] L. J. Templin, *Reactor Physics Constants*, 2nd Edition, ANL-5800, (Argonne National Laboratory, Argonne, IL, 1963).

[18] G. F. Knoll, *Radiation Detection and Measurement*, 3rd Edition, (John Wiley & Sons, New York, 2000).

APPENDIX A

LND Model 252 ^{3}He Cylindrical Neutron Detector Physical Specifications



APPENDIX B

Spectron Model SH 50056-A-003 Ceramic Tilt Switch

Technical Specifications

Characteristics	Model SH50056-A-003
Linear Range (arc degrees)	3
Total Range (arc degrees)	6
Output (mV/arc degree)	0.4
Resolution (arc degrees)	< 0.0003
Null Repeatability (arc degrees)	< 0.0008
Symmetry at ½ Linear Scale (%)	< 2
Accuracy at ½ Linear Scale (% Full Scale)	< 2
Accuracy at Full Scale (% Full Scale)	< 8
Null Impedance (kΩ) ± 20%	12
Null Stability, 12 hours at 25°C (arc degrees)	< 0.0005
Shock Survival (G's)	100
Operating Temperature (°C)	-54 to +125
Storage Temperature (°C)	-76 to +150
Temperature Coefficient of Scale (%/°C)	0.6

APPENDIX C

C++ Delayed Neutron Program Source Code

delay_3s.c

```
/*************************************************************************
*
* Name:  delay_3s
* Arguments:  ---
* Returns:  ---
*
* Delay for 3 seconds.
*
*************************************************************************/
#include "main.h"
extern int ULStat, BoardNum, LoadValue, RegName;
extern WORD delay_count;
extern int CounterNum;
void delay_3s (void)
{
    RegName=LOADREG2;
    ULStat = cbCLoad (BoardNum, RegName, LoadValue);
    delay_count=0;
    while (delay_count!=8536)
    {
        CounterNum=2;
        ULStat = cbCIn (BoardNum, CounterNum, &delay_count);
    }
}
```

execute_menu.c

```
/*************************************************************************
*
* Name:  execute_menu
* Arguments:  ---
* Returns:  ---
*
*************************************************************************/
#include "main.h"
#include "execute_menu.h"
int x = 0;
int execute_menu (void)
```

```
//   main_menu --> Selection 1 - Running the system
{
    if (flag == 1)
    {
        clear_screen ();
        result_menu ();
        flag = 0;
    }
    printf("Press 1 and then Enter key to shoot the sample.\n");
    printf("Press 2 and then Enter key to return to main menu.\n");
    for (i=0;i<41;++i)
    selection[i] = 0;
    i=0;
    /* Clears input buffer for getchar() */
    if (x == 1)
    {
        ch = getchar();
        x = 0;
    }
    while ((ch = getchar()) != '\n')
    {
        selection[i++] = ch;
        x++;
    }
/*  if (flag == 1)
    {
        flag = 0;
        result_menu ();
    }
*/
    if((selection[0] == '1' && selection[1] == 0))
    //      shoots sample
    {
        //clear_screen ();
        flag=0;
        //When execution is finished, will go to result_menu
        printf("Enter irradiation time in seconds --> ");
        scanf("%d", &irradiation_time);
        printf("\nEnter reading time for the\n     detector in seconds ");
        printf("and press\n     Enter to execute the program --> ");
        scanf("%d", &detector_time);
        clear_screen ();
        //core_init_count=0; core_final_count=0;
        detector_init_count=0;
```

```
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//detector_final_count=0; flag_core=0; flag_detector=0;
RegName = LOADREG1;
ULStat = cbCLoad (BoardNum, RegName, LoadValue);
//Initialize load value
core_sample_inside_flag = 0;
detector_sample_pass_flag = 0;
core_no_sensor_flag = 0;
detector_no_sensor_flag = 0;
loadreg2_flag=0;
loadreg3_flag=0;
loadreg4_flag=0;
core_init_count=0;
detector_init_count=0;
core_final_count=0;
detector_final_count=0;
flag_core=0;
detector_sensor_trigger=0;
two_sensor_execution_flag=0;
one_sensor_execution_flag=0;
no_sensor_execution_flag=0;
min=0;
sec=0;
millisec=0;
while (two_sensor_execution_flag != 5)
{
        //Parameters:
        //BoardNum    :the number used by CB.CFG to describe this board
        //CounterNum  :the counter to be setup
        //count       :the count value in the counter
        CounterNum = 1;
        ULStat = cbCIn (BoardNum, CounterNum, &count);
        //First: Check to see if current_value is equal to count
        //If so, current_value is still count value and
        //no change has occured; therefore, do not operate
        //on the no change.
        //
        //If count has changed, values do not equal and function call
        //occurs in response to new count value. The new value will
        //be reflected in the current_value variable for use in the first step.
        if(!(current_value == count))
        {
                master_clock_conversion ();
                if(flag_sensor == 3)
```

```
                        //using both detector sensor and core sensor
                        {
                                two_sensor_execution ();
                                if(two_sensor_execution_flag == 5)
                                {
                                        flag = 1;
                                }
                        }
                        else if(flag_sensor == 2)
                        //using only detector sensor
                        {
                                one_sensor_execution ();
                                if(one_sensor_execution_flag == 5)
                                {
                                        flag = 1;
                                }
                        }
                        else if (flag_sensor == 1) // using no sensor
                        {
                                no_sensor_execution ();
                                if(no_sensor_execution_flag == 5)
                                {
                                        flag = 1;
                                }
                        }
                        else
                        {
                                printf("break out!!!!!!\n\n");
                                break;
                        }
                }
        current_value = count;
        }
}
else if((selection[0] == '2' && selection[1] == 0))
//return to main_menu
{
        clear_screen ();
        main_menu ();
}
/*      else
{
        clear_screen ();
        flag=0;
```

```
            printf("Value input was incorect, please re-enter the correct selection.\n\n");
            execute_menu ();
    }
    */
    execute_menu ();
    return 0;
}
```

execute_menu.h

```
extern WORD count;
extern WORD bit_value;
extern WORD delay_count;
extern int BoardNum;
extern int bit_num;
extern long core_init_count;
extern long core_final_count;
extern int core_no_sensor_flag;
extern int core_sample_inside_flag;
extern int core_sensor_trigger;
extern int CounterNum;
extern int current_value;
extern long detector_init_count;
extern long detector_final_count;
extern int detector_no_sensor_flag;
extern int detector_sample_pass_flag;
extern int detector_sensor_trigger;
extern int detector_time;
extern int two_sensor_execution_flag;
extern int one_sensor_execution_flag;
extern int no_sensor_execution_flag;
extern int flag;
extern int flag_core;
extern int flag_detector;
extern int flag_sensor;
extern int function;
extern int irradiation_time;
extern int loadreg2_flag;
extern int loadreg3_flag;
extern int loadreg4_flag;
extern int LoadValue;
extern long millisec_total;
extern int port_type;
extern int RegName;
```

```
extern int shoot_solenoid_1_flag;
extern int ULStat;
extern char selection[40];
extern char ch;
//character
extern int i;
//position in the array
extern int min;
extern int sec;
extern int millisec;
```

exit_menu.c

```
/*****************************************************************************
*
* Name:  exit_menu
* Arguments:  ---
* Returns:  ---
*
* To exit the program completely.
*
*****************************************************************************/
#include "main.h"
void exit_menu (void)
//    main_menu --> Selection 3 - Exiting the system
{
     exit(0);
}
```

initialize.c

```
/*****************************************************************************
*
* Name:  initialize
* Arguments:  ---
* Returns:  ---
*
* Intitializing the PCI-CTR05 board and the variables.
*
*****************************************************************************/
#include "main.h"
#include "initialize.h"
void initialize (void)
{
```

```
//close all the solenoids
ULStat = cbDOut(1,AUXPORT,0);
//Declare UL Revision Level
ULStat = cbDeclareRevision(&RevLevel);
//Initiate error handling
//Parameters:
//PRINTALL :all warnings and errors encountered will be printed
//DONTSTOP :program will continue even if error occurs.
//Note that STOPALL and STOPFATAL are only effective in
//Windows applications, not Console applications.
cbErrHandling (PRINTALL, DONTSTOP);
//SET UP THE DISPLAY SCREEN
//Initialize the board level features
//Parameters:
//BoardNum    :the number used by CB.CFG to describe this board
//ChipNum     :chip being initialized (1 for CTR5, 1 or 2 for CTR10
//FOutDivider :the F-Out divider (0-15)
//FOutSource  :the signal source for F-Out
//Compare1    :status of comparator 1
//Compare2    :status of comparator 2
//TimeOfDay   :time of day control mode
ChipNum = 1;
FOutDivider = 1;
FOutSource = FREQ4;
Compare1 = DISABLED;
Compare2 = DISABLED;
TimeOfDay = DISABLED;
//TimeOfDay = 3;
ULStat = cbC9513Init (BoardNum, ChipNum, FOutDivider, \
FOutSource,Compare1,Compare2, TimeOfDay);
//Set the configurable operations of the counter
//Parameters:
//BoardNum       :the number used by CB.CFG to describe this board
//CounterNum     :the counter to be configured (0-5)
//GateControl    :gate control value
//CounterEdge    :which edge to count
//CountSource    :signal source
//SpecialGate    :status of special gate
//ReLoad         :method of reloading the counter
//RecycleMode    :recycle mode
//BCDMode        :counting mode, BCD or binary
//CountDirection :direction for the counting (COUNTUP or
//COUNTDOWN)
//OutputControl  :output signal type and level
```

```
GateControl = NOGATE;
CounterEdge = POSITIVEEDGE;
CountSource = FREQ4;
SpecialGate = DISABLED;
ReLoad = LOADREG;
//ReLoad = LOADANDHOLDREG;
RecycleMode = RECYCLE;
//RecycleMode = ONETIME;
BCDMode = DISABLED;
CountDirection = COUNTUP;
OutputControl = ALWAYSLOW;
//Initialize Counter 1
CounterNum = 1;
ULStat = cbC9513Config (BoardNum, CounterNum , GateControl, \
        CounterEdge, CountSource, SpecialGate, ReLoad,\
        RecycleMode, BCDMode, CountDirection, OutputControl);
//Initialize Counter 2
CounterNum = 2;
ULStat = cbC9513Config (BoardNum, CounterNum , GateControl, \
        CounterEdge, CountSource, SpecialGate, ReLoad, \
        RecycleMode, BCDMode, CountDirection, OutputControl);
//Initialize Counter 3
CounterNum = 3;
ULStat = cbC9513Config (BoardNum, CounterNum , GateControl, \
        CounterEdge, CountSource, SpecialGate, ReLoad, \
        RecycleMode, CDMode, CountDirection, OutputControl);
//Initialize Counter 4
CounterNum = 4;
ULStat = cbC9513Config (BoardNum, CounterNum , GateControl, \
        CounterEdge, CountSource, SpecialGate, ReLoad, \
        RecycleMode, BCDMode, CountDirection, OutputControl);
//Initialize Counter 5
CounterNum = 5;
ULStat = cbC9513Config (BoardNum, CounterNum , GateControl, \
        CounterEdge, CountSource, SpecialGate, ReLoad, \
        RecycleMode, BCDMode, CountDirection, OutputControl);
//Send a starting value to the counter with cbCLoad()
//Parameters:
//BoardNum    :the number used by CB.CFG to describe this board
//RegName     :the counter to be loading with the starting value
//LoadValue   :the starting value to place in the counter
LoadValue = 5536;
RegName = LOADREG1;
ULStat = cbCLoad (BoardNum, RegName, LoadValue);
```

```
    //Initial current_value to 0
    current_value = 0;
    //Use a loop to keep checking the counter value with cbCIn()
    GetTextCursor (&Col, &Row);
    close_all_solenoid ();
}
```

initialize.h

```
extern int ULStat;
//extern float RevLevel = (extern float)CURRENTREVNUM;
extern float RevLevel;
extern int ChipNum;
extern int FOutDivider;
extern int FOutSource;
extern int Compare1;
extern int Compare2;
extern int TimeOfDay;
extern int BoardNum;
extern int GateControl;
extern int CounterEdge;
extern int CountSource;
extern int SpecialGate;
extern int ReLoad;
extern int RecycleMode;
extern int BCDMode;
extern int CountDirection;
extern int OutputControl;
extern int CounterNum;
extern int LoadValue;
extern int RegName;
extern int current_value;
extern int Col;
extern int Row;
```

main_.c

```
/*main.c*************************************************************
File:  main.c
Library Call Demonstrated:  9513 Counter Functions
    cbC9513Init()
    cbC9513Config()
    cbCLoad()
    cbCIn()
Purpose:  Operate the counter.
```

```
Demonstration:  Initializes, configures, loads, and
                reads the counter.
Other Library Calls:  cbErrHandling()
Special Requirements:  Board 0 must have a 9513 Counter.
                       Program uses the internal clock to count.
Copyright (c) 1995-2002, Measurement Computing Corp.
All Rights Reserved.
*********************************************************************
#include "main.h"
#include "variables.h"
/*********************************************************************
*
* Name:  main
* Arguments:  ---
* Returns:  ---
*
* Main function
*
*********************************************************************/
void main()
{
    initialize ();
/*
    for (bit_num=0; bit_num<8; bit_num++)
    {
        bit_num=3;
        ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
        printf("%d ", bit_value);
    }
*/
    bit_num=3;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    if (bit_value == 1)
    //permit reading
    {
        printf("You NEED the PERMIT to run the system!\n");
        main ();
    }
    else
    {
        clear_screen ();
        sensor_menu ();
        //Sensor selection
        ULStat = cbDIn (BoardNum, port_type, &bit_value);
```

```
            if (flag_sensor == 1 && bit_value > 0xF7)
            printf("Check the sensors or power -- No sensors selection.\n");
            else if ((flag_sensor == 2 && bit_value > 0xF3))
            printf("Check the sensors or power -- One sensor selection.\n");
            else if (flag_sensor == 3 && bit_value == 0xF3)
            printf("Check the sensors or power -- Two sensors selection.\n");
            else
            {
                  while (1)
                  {
                        clear_screen ();
                        main_menu ();
                  }
            }
            exit(0);
      }
 }
/*************************************************************************
*
* Name:  shoot_solenoid_1
* Arguments:  ---
* Returns:  ---
*
* Shoots Solenoid 1 for 3 seconds
*
*************************************************************************/
void shoot_solenoid_1 (void)
{
     printf("Solenoid 1 fired!\n\n");
     ULStat = cbDOut(BoardNum =0,AUXPORT,1);
     delay_3s ();
ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}
/*************************************************************************
*
* Name:  shoot_solenoid_2
* Arguments:  ---
* Returns:  ---
*
* Shoots Solenoid 2 for 3 seconds
*
*************************************************************************/
void shoot_solenoid_2 (void)
{
```

```
    printf("Solenoid 2 fired!\n\n");
    ULStat = cbDOut(BoardNum =0,AUXPORT,2);
    delay_3s ();
    ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}
/*************************************************************************
*
* Name:  shoot_solenoid_3
* Arguments:  ---
* Returns:  ---
*
* Shoots Solenoid 3 for 3 seconds
*
*************************************************************************/
void shoot_solenoid_3 (void)
{
    printf("Solenoid 3 fired!\n\n");
    ULStat = cbDOut(BoardNum =0,AUXPORT,4);
    delay_3s ();
    ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}
/*************************************************************************
*
* Name:  shoot_solenoid_4
* Arguments:  ---
* Returns:  ---
*
* Shoots Solenoid 1 for 3 seconds
*
*************************************************************************/
void shoot_solenoid_4 (void)
{
    printf("Solenoid 4 fired!\n\n");
    ULStat = cbDOut(BoardNum =0,AUXPORT,8);
    delay_3s ();
    ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}
/*************************************************************************
*
* Name:  close_all_solenoid
* Arguments:  ---
* Returns:  ---
*
* Close all solenoids
```

```
*
*************************************************************************/
void close_all_solenoid(void)
{
    ULStat = cbDOut(BoardNum =0,AUXPORT,0);
}
/*************************************************************************
*
* Name:  clear_screen
* Arguments:  ---
* Returns:  ---
*
* Main Counter
*
*************************************************************************/
#define BIOS_VIDEO   0x10
void clear_screen (void)
{
    COORD coordOrg = {0, 0};
    DWORD dwWritten = 0;
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (INVALID_HANDLE_VALUE != hConsole)
    {
        FillConsoleOutputCharacter(hConsole, ' ', 80 * 50, coordOrg, \
            &dwWritten);
}
    MoveCursor(0, 0);
return;
/*************************************************************************
*
* Name:  MoveCursor
* Arguments:  x,y - screen coordinates of new cursor position
* Returns:  ---
*
* Positions the cursor on screen.
*
*************************************************************************/
void MoveCursor (int x, int y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (INVALID_HANDLE_VALUE != hConsole)
    {
        COORD coordCursor;
        coordCursor.X = (short)x;
```

```
            coordCursor.Y = (short)y;
            SetConsoleCursorPosition(hConsole, coordCursor);
    }
return;
}
/**************************************************************************
*
* Name:  GetTextCursor
* Arguments:  x,y - screen coordinates of new cursor position
* Returns:  *x and *y
*
* Returns the current (text) cursor position.
*
**************************************************************************/
void GetTextCursor (int *x, int *y)
{
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_SCREEN_BUFFER_INFO csbi;
    *x = -1;
    *y = -1;
    if (INVALID_HANDLE_VALUE != hConsole)
    {
            GetConsoleScreenBufferInfo(hConsole, &csbi);
            *x = csbi.dwCursorPosition.X;
            *y = csbi.dwCursorPosition.Y;
    }
return;
}
```

main.h

```
/* Include files */
#include <windows.h>
#include <stdio.h>
#include <conio.h>
#include "cbw.h"
/* Prototypes*/
void clear_screen (void);
void close_all_solenoid (void);
void sensor_core (void);
void delay_3s (void);
void sensor_detector (void);
int execute_menu (void);
void exit_menu (void);
```

```
void GetTextCursor (int *x, int *y);
void initialize (void);
int main_menu (void);
void master_clock_conversion (void);
void MoveCursor (int x, int y);
void no_sensor (void);
void result_menu (void);
void sensor_menu (void);
void shoot_solenoid_1 (void);
void shoot_solenoid_2 (void);
void shoot_solenoid_3 (void);
void shoot_solenoid_4 (void);
void test_menu (void);
void two_sensor_execution (void);
void one_sensor_execution (void);
void no_sensor_execution (void);
```

main_menu.c

```
/*************************************************************************
*
* Name:  main_menu
* Arguments:  ---
* Returns:  ---
*
* Displays the main menu to select sub-menus to:
*    -- Run the program.
*    -- View the previous results from the last execution.
*    -- Test the solenoids.
*    -- Exit the program.
*
*************************************************************************/
#include "main.h"
extern char selection[40];
extern char ch;
//    character
extern int i;
//    position in the array
extern int flag;
int main_menu (void)
{
     flag=0;
     printf("Delayed Neutron Program\n");
     printf("Select an option below:\n\n");
```

```
printf("1. Run the delayed neutron system.\n");
printf("2  VIEW the RESULTS.\n");
printf("3. Test solenoids.\n");
printf("4. EXIT\n");
for (i=0;i<41;++i)
selection[i] = 0;
i=0;
while ((ch = getchar()) != '\n')
        selection[i++] = ch;
if(selection[0] == '1' && selection[1] == 0)
//      execute_menu
{
        clear_screen ();
        execute_menu ();
}
else if(selection[0] == '2' && selection[1] == 0)
{
        clear_screen ();
        result_menu ();
}
else if(selection[0] == '3' && selection[1] == 0)
//      test_menu
{
        clear_screen ();
        test_menu ();
}
else if(selection[0] == '4' && selection[1] == 0)
//      exit_menu
{
exit_menu ();
}
Else
{
        clear_screen ();
        printf("Value input was incorect, please re-enter the correct selection.\n\n");
}
main_menu ();
return 0;
}
```

master_clock_conversion.c

```
/*************************************************************************
*
```

```
* Name:  master_clock_conversion
* Arguments:  ---
* Returns:  ---
*
* Main Clock.
*
*****************************************************************************/
#include "main.h"
#include "master_clock_conversion.h"
void master_clock_conversion (void)
{
    //MoveCursor (Col, Row + 0);
    MoveCursor (0, 0);
    //printf ("The value of Counter %u is %u     \n", CounterNum, count);
    sec = (count-5536)/1000;
    sec_total = 60*min+sec;
    millisec = count-1000*sec-5536;
    millisec_total = sec_total*1000+millisec;
    sec1 = sec%10;
    sec2 = (sec-sec%10)/10;
    min1 = min%10;
    min2 = (min-min%10)/10;
    millisec1 = (millisec%100)%10;
    millisec2 = ((millisec-millisec1)/100)%10;
    millisec3 = (millisec-millisec1-10*millisec2)/100;
    if (count==65535)
    {
        ++min;
        clear_screen ();
    }
    printf("Master Clock: %d%d:%d%d:%d%d%d\n", min2, min1, sec2, sec1, \
        millisec3, millisec2, millisec1);
    //printf("sec_total = %d\n", sec_total);
    //printf("millisec_total = %d\n", millisec_total);
    //printf("core_sensor_trigger = %d\n", core_sensor_trigger);
    //printf("detector_sensor_trigger = %d\n", detector_sensor_trigger);
    //printf("core_init_count = %d\n", core_init_count);
    //printf("core_final_count = %d\n", core_final_count);
    //printf("detector_init_count = %d\n\n", detector_init_count);
}
```

master_clock_conversion.h

```
extern long core_final_count;
extern long core_init_count;
extern int core_sensor_trigger;
extern int CounterNum;
extern WORD count;
extern long detector_init_count;
extern long detector_sensor_trigger;
extern int sec;
extern int sec1;
extern int sec2;
extern int sec_total;
extern int min;
extern int min1;
extern int min2;
extern int millisec;
extern int millisec1;
extern int millisec2;
extern int millisec3;
extern long millisec_total;
```

no_sensor_execution.c

```
#include "main.h"
#include "no_sensor_execution.h"
void no_sensor_execution (void)
{
    if (no_sensor_execution_flag == 0)
    {
        no_sensor_execution_flag++;
        printf("Solenoid 1 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,1);
        //Solenoid 1 fired
        core_init_count = millisec_total;
    }
    if ((no_sensor_execution_flag == 1) && (core_init_count + 1000 == millisec_total))
    {
        no_sensor_execution_flag++;
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        core_init_count = millisec_total;
        flag_core = irradiation_time*1000 + core_init_count;
    }
    if (flag_core == millisec_total && no_sensor_execution_flag == 2)
    {
        no_sensor_execution_flag++;
```

```
            printf("Solenoid 2 fired!\n\n");
            ULStat = cbDOut(BoardNum =0,AUXPORT,2);//shoot solenoid 2
            core_final_count = millisec_total;
        }
        if ((no_sensor_execution_flag == 3)&& (core_final_count + 1000 == millisec_total))
        {
            no_sensor_execution_flag++;
            ULStat = cbDOut(BoardNum =0,AUXPORT,0);
            detector_init_count = millisec_total;
            flag_detector = detector_time*1000 + detector_init_count;
        }
        if (flag_detector == millisec_total && no_sensor_execution_flag ==4)
        {
            no_sensor_execution_flag++;
            detector_final_count = millisec_total;
            shoot_solenoid_3 ();
        }
}
```

no_sensor_exeuction.h

```
extern int no_sensor_execution_flag;
extern int ULStat;
extern int BoardNum;
extern int core_sensor_trigger;
extern long core_init_count;
extern WORD count;
extern int flag_core;
extern int irradiation_time;
extern long core_final_count;
extern int detector_sensor_trigger;
extern int detector_sample_pass_flag;
extern long detector_init_count;
extern int flag_detector;
extern int detector_time;
extern long detector_final_count;
extern int bit_num;
extern int port_type;
extern long millisec_total;
extern WORD bit_value;
```

one_sensor_execution.c

```
#include "main.h"
```

```
#include "one_sensor_execution.h"
void one_sensor_execution (void)
{
    if (one_sensor_execution_flag == 0)
    {
        one_sensor_execution_flag++;
        printf("Solenoid 1 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,1);
        //Solenoid 1 fired
        core_init_count = millisec_total;
    }
    if ((one_sensor_execution_flag == 1) && (core_init_count + 1000 == \
        millisec_total))
    {
        one_sensor_execution_flag++;
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        //Solenoids off
        core_init_count = millisec_total;
        flag_core = irradiation_time*1000 + core_init_count;
    }
    if ((flag_core == millisec_total) && (one_sensor_execution_flag == 2))
    {
        one_sensor_execution_flag++;
        printf("Solenoid 2 fired!\n\n");
        ULStat = cbDOut(BoardNum =0,AUXPORT,2);//shoot solenoid 2
        core_final_count = millisec_total;
    }
    //Read the value off the sensor in the detector
    bit_num=2;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    detector_sensor_trigger = bit_value;
    if (detector_sensor_trigger == 1 && detector_sample_pass_flag == 0 && \
        one_sensor_execution_flag == 3)
    {
        one_sensor_execution_flag++;
        detector_sample_pass_flag++;
        detector_init_count = millisec_total;
        //Timestamp - record time sample arrived in the detector
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        flag_detector = detector_time*1000 + detector_init_count;
    }
    if (flag_detector == millisec_total && one_sensor_execution_flag == 4)
    {
        one_sensor_execution_flag++;
```

```
            detector_final_count = millisec_total;
            //Timestamp - record time sample left the detector
            shoot_solenoid_3 ();
    }
}
```

one_sensor_execution.h

```
extern int one_sensor_execution_flag;
extern int ULStat;
extern int BoardNum;
extern int core_sensor_trigger;
extern long core_init_count;
extern WORD count;
extern int flag_core;
extern int irradiation_time;
extern long core_final_count;
extern int detector_sensor_trigger;
extern int detector_sample_pass_flag;
extern long detector_init_count;
extern int flag_detector;
extern int detector_time;
extern long detector_final_count;
extern int bit_num;
extern int port_type;
extern WORD bit_value;
extern long millisec_total;
```

result_menu.c

```
/*****************************************************************************
*
* Name:  result_menu
* Arguments:  ---
* Returns:  ---
*
*****************************************************************************/
#include "main.h"
#include "result_menu.h"
void result_menu (void)
{
    flag=1;
    printf("The results are:\n");
    //**************  time_irradiation ******************
```

```
    time_irradiation = core_final_count - core_init_count;
    min = time_irradiation/60000;
    sec = (time_irradiation-min*60000)/1000;
    millisec = time_irradiation-1000*sec-60000*min;
    sec1 = sec%10;
    sec2 = (sec-sec%10)/10;
    min1 = min%10;
    min2 = (min-min%10)/10;
    millisec1 = (millisec%100)%10;
    millisec2 = ((millisec-millisec1)/100)%10;
    millisec3 = (millisec-millisec1-10*millisec2)/100;
    printf("\tIrradiation time:     %d%d:%d%d:%d%d%d (min:sec:milli-sec)\n", \
          min2, min1 , sec2, sec1, millisec3, millisec2, millisec1);
    //***************  fly_time ******************
    fly_time = detector_init_count - core_final_count;
    min = fly_time/60000;
    sec = (fly_time-min*60000)/1000;
    millisec = fly_time-1000*sec-60000*min;
    sec1 = sec%10;
    sec2 = (sec-sec%10)/10;
    min1 = min%10;
    min2 = (min-min%10)/10;
    millisec1 = (millisec%100)%10;
    millisec2 = ((millisec-millisec1)/100)%10;
    millisec3 = (millisec-millisec1-10*millisec2)/100;
    printf("\tFly time:             %d%d:%d%d:%d%d%d (min:sec:milli-sec)\n", \
          min2, min1 , sec2, sec1, millisec3, millisec2, millisec1);
    //*************** time_detection ******************
    time_detection = detector_final_count - detector_init_count;
    //time_detection
    min = time_detection/60000;
    sec = (time_detection-min*60000)/1000;
    millisec = time_detection-1000*sec-60000*min;
    sec1 = sec%10;
    sec2 = (sec-sec%10)/10;
    min1 = min%10;
    min2 = (min-min%10)/10;
    millisec1 = (millisec%100)%10;
    millisec2 = ((millisec-millisec1)/100)%10;
    millisec3 = (millisec-millisec1-10*millisec2)/100;
    printf("\tTime detection:       %d%d:%d%d:%d%d%d (min:sec:milli-sec)\n\n", \
          min2, min1 , sec2, sec1, millisec3, millisec2, millisec1);
}
```

result_menu.h

```
extern int flag;
extern int core_init_count;
extern int core_final_count;
extern int detector_init_count;
extern int detector_final_count;
extern int detector_time;
extern int irradiation_time;
extern int time_irradiation;
extern int fly_time;
extern int time_detection;
extern int sec;
extern int sec1;
extern int sec2;
extern int min;
extern int min1;
extern int min2;
extern int millisec;
extern int millisec1;
extern int millisec2;
extern int millisec3;
```

sensor_core.c

```
/*********************************************************************
*
* Name:  sensor_core
* Arguments:  ---
* Returns:  ---
*
* Timestamps for core sensor and shoots Solenoid 2
*
*********************************************************************/
#include "main.h"
extern int core_sample_inside_flag, core_sensor_trigger, core_init_count;
extern int count, flag_core, irradiation_time, core_final_count;
void sensor_core (void)
{
    if (core_sample_inside_flag == 0 && core_sensor_trigger == 1)
    {
        //Timestamp - record time sample arrived in the core
        core_sample_inside_flag = 1;
        core_init_count = count;
```

```
            flag_core = irradiation_time*1000 + core_init_count;
    }
    else if ((flag_core == count) && (core_sensor_trigger == 1))
    {
            shoot_solenoid_2 ();
            //printf("Fired Solenoid 2\n");
            //Timestamp - record time sample left the core
            core_final_count = count;
    }
}
```

sensor_detector.c

```
/*********************************************************************
*
* Name:  sensor_detector
* Arguments:  ---
* Returns:  ---
*
* Timestamps for detector sensor and shoots Solenoid 3
*
*********************************************************************/
#include "main.h"
#include "sensor_detector.h"
void sensor_detector (void)
{
    bit_num=2;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    detector_sensor_trigger = bit_value;
    if (detector_sensor_trigger == 1 && detector_sample_pass_flag == 0)
    {
            detector_sample_pass_flag++;
            //Timestamp - record time sample arrived in the detector
            detector_init_count = count;
            flag_detector = detector_time*1000 + detector_init_count;
    }
    else if (flag_detector == count)
    {
            shoot_solenoid_3 ();
            //printf("Fired Solenoid 3\n");
            //Timestamp - record time sample left the detector
            detector_final_count = count;
    }
}
```

sensor_detector.h

```
extern unsigned short bit_value;
extern int bit_num;
extern int ULStat;
extern int BoardNum;
extern int port_type;
extern int function;
extern int detector_sensor_trigger;
extern int detector_sample_pass_flag;
extern int detector_init_count;
extern int count;
extern int flag_detector;
extern int detector_time;
extern int detector_final_count;
```

sensor_menu.c

```
/*****************************************************************************
*
* Name:  sensor_menu
* Arguments:  ---
* Returns:  ---
*
* Displays the sensor menu to select sub-menus to:
*    -- Run the program using no sensors.
*    -- Run the program using one sensor located with the detectors.
*    -- Run the program using two sensors located with the detectors and
*                                                                    another
sensor located in the core.
*
*****************************************************************************/
#include "main.h"
//#include "external_variables.h"
extern char selection[40];
extern char ch;
//    character
extern int i;
//    position in the array
extern int flag_sensor;
void sensor_menu (void)
{
     printf("Select the correct option below that corresponds with your hardware   \
          setup:\n\n");
```

```
    printf("Enter 1 -- not using any sensors.\n");
    printf("Enter 2 -- using one sensor located with the detectors.\n");
    printf("Enter 3 -- using two sensors, one located with the\n");
    printf("           detectors and one located at the core.\n");
    for (i=0;i<41;++i)
        selection[i] = 0;
    i=0;
    while ((ch = getchar()) != '\n')
        selection[i++] = ch;
        //      Error Correction
        if (((selection[0] != '1') && (selection[0] != '2') && (selection[0] != '3')) || \
            selection[1] != 0)
        {
            clear_screen ();
            //printf("i = %d selections %d %d %d %d %d %d\n", i, \
            //      selection[0], selection[1], selection[2], selection[3], \
            //      selection[4], selection[5]);
            printf("Value input was incorect, please re-enter the correct \
                selection.\n\n");
            sensor_menu ();
    }
    flag_sensor = selection[0] - 0x30;
    printf("flag sensor = %d\n", flag_sensor);
}
```

test_menu.c

```
/*****************************************************************************
*
* Name:  test_menu
* Arguments:  ---
* Returns:  ---
*
*****************************************************************************/
#include "main.h"
extern char selection[40];
extern char ch;
//character
extern int i;
//position in the array
void test_menu (void)
//main_menu --> Selection 2 - Testing the solenoids
{
    //Initializes Solenoids to be closed
```

```
close_all_solenoid ();
printf("Press 1 and then Enter key to activate Solenoid 1.\n");
printf("Press 2 and then Enter key to activate Solenoid 2.\n");
printf("Press 3 and then Enter key to activate Solenoid 3.\n");
printf("Press 4 and then Enter key to activate Solenoid 4.\n");
printf("Press 5 and then Enter key to return to main menu.\n");
for (i=0;i<41;++i)
selection[i] = 0;
i=0;
while ((ch = getchar()) != '\n')
        selection[i++] = ch;
if(selection[0] == '1' && selection[1] == 0)
//shoots sample --> Dout1 using decimal value 1
{
        clear_screen ();
        shoot_solenoid_1 ();
        test_menu ();
}
else if(selection[0] == '2' && selection[1] == 0)
{
        clear_screen ();
        shoot_solenoid_2 ();
        test_menu ();
}
else if(selection[0] == '3' && selection[1] == 0)
{
        clear_screen ();
        shoot_solenoid_3 ();
        test_menu ();
}
else if(selection[0] == '4' && selection[1] == 0)
{
        clear_screen ();
        shoot_solenoid_4 ();
        test_menu ();
}
else if(selection[0] == '5' && selection[1] == 0)
{
        clear_screen ();
        main_menu ();
}
else
//Wrong selection, re-enter the selection
{
```

```
            clear_screen ();
            printf("Value input was incorect, please re-enter the correct selection.\n\n");
            test_menu ();
    }
}
```

two_sensor_execution.c

```
#include "main.h"
#include "two_sensor_execution.h"
void two_sensor_execution (void)
{
    if (two_sensor_execution_flag == 0)
    {
            two_sensor_execution_flag++;
            //printf("Solenoid 1 fired!\n\n");
            //$$$$$$$$$$$$$$$$$$$$$$$$
            //ULStat = cbDOut(BoardNum =0,AUXPORT,1);
            //Solenoid 1 fired  $$$$$$$$$$$$$$$$$$$$$$$$
            //shoot_solenoid_1 ();
    }
    //Read the value of the sensor in the core
    //while (two_sensor_execution_flag == 1 && core_sensor_trigger == 0)
    //********************** waiting until sample enters the core
    //{
            //**********************
            bit_num=1;
            ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
            core_sensor_trigger = bit_value;
    //}
    //**********************
    if (two_sensor_execution_flag == 1 && core_sensor_trigger==1)
    {
            two_sensor_execution_flag++;
            //ULStat = cbDOut(BoardNum =0,AUXPORT,0);
            //Solenoids off
            //$$$$$$$$$$$$$$$$$$$$$$$$
            core_init_count = millisec_total;
            //The time when the sample enters the core
            flag_core = irradiation_time*1000 + core_init_count;
            //Adding time the user entered for sample to be left in
            //the core with the timestamp of the current time, which
            //creates a result that the sample should leave out of
            //the core.
```

```
    }
    if ((flag_core == millisec_total) && (two_sensor_execution_flag == 2))
    {
        two_sensor_execution_flag++;
        core_final_count = millisec_total;
        //Timestamp - record time sample left the core
        printf("Solenoid 4 fired!\n\n");
        //$$$$$$$$$$$$$$$$$$$$$$$$
        ULStat = cbDOut(BoardNum =0,AUXPORT,8);
        //Solenoid 4 fired  $$$$$$$$$$$$$$$$$$$$$$$$
    }
    //Read the value of the sensor in the detector
    bit_num=2;
    ULStat = cbDBitIn (BoardNum, port_type, bit_num, &bit_value);
    detector_sensor_trigger = bit_value;
    if (detector_sensor_trigger == 1 && detector_sample_pass_flag == 0 && \
        two_sensor_execution_flag ==3)
    {
        two_sensor_execution_flag++;
        detector_sample_pass_flag++;
        detector_init_count = millisec_total;
        //Timestamp - record time sample arrived in the detector
        ULStat = cbDOut(BoardNum =0,AUXPORT,0);
        //Solenoids off
        flag_detector = detector_time*1000 + detector_init_count;
    }
    if (flag_detector == millisec_total && two_sensor_execution_flag == 4)
    {
        two_sensor_execution_flag++;
        shoot_solenoid_3 ();
        detector_final_count = millisec_total;
        //Timestamp - record time sample left the detector
    }
}
```

two_sensor execution.h

```
extern int two_sensor_execution_flag;
extern int ULStat;
extern int BoardNum;
extern int core_sensor_trigger;
extern long core_init_count;
extern WORD count;
extern int flag_core;
```

```
extern int irradiation_time;
extern long core_final_count;
extern int detector_sensor_trigger;
extern int detector_sample_pass_flag;
extern long detector_init_count;
extern int flag_detector;
extern int detector_time;
extern long detector_final_count;
extern int bit_num;
extern int port_type;
extern unsigned short bit_value;
extern long millisec_total;
```

variables.h

```
/*   Variable Declarations */
//PCI-CTR05
int Row, Col;
int ULStat = 0;
int BoardNum = 0;
int ChipNum, FOutDivider, FOutSource, Compare1, Compare2, TimeOfDay;
int CounterNum, GateControl, CounterEdge, CountSource, SpecialGate;
int ReLoad, RecycleMode, BCDMode, CountDirection, OutputControl;
int LoadValue;
int RegName;
int port_type=AUXPORT;
int bit_num;
//master_clock_conversion
int sec=0, sec1=0, sec2=0, min=0, min1=0, min2=0, sec_total=0,
sec_total_current_value=0;
int millisec=0, millisec3=0, millisec2=0, millisec1=0;
long millisec_total=0;
int irradiation_time_min2_temp, irradiation_time_min1_temp;
int  irradiation_time_sec2_temp, irradiation_time_sec1_temp;
int irradiation_time_millisec3_temp, irradiation_time_millisec2_temp,
irradiation_time_millisec1_temp;
int detector_time_min2_temp, detector_time_min1_temp;
int  detector_time_sec2_temp, detector_time_sec1_temp;
int detector_time_millisec3_temp, detector_time_millisec2_temp,
detector_time_millisec1_temp;
//flags
int flag=0, flag_sensor=0, flag_core, flag_detector, core_sample_inside_flag;
int shoot_solenoid_1_flag=0;
int core_no_sensor_flag = 0, detector_no_sensor_flag = 0;
```

```
int core_sensor_trigger;
int detector_sensor_trigger, detector_sample_pass_flag;
long detector_init_count, detector_final_count,core_init_count, core_final_count;
int irradiation_time, detector_time;
int loadreg2_flag, loadreg3_flag, loadreg4_flag;
WORD count, delay_count, data, bit_value, delay_count1;
float RevLevel = (float)CURRENTREVNUM;
int current_value;
int two_sensor_execution_flag;
int one_sensor_execution_flag;
int no_sensor_execution_flag;
int time_irradiation;
int fly_time;
int time_detection;
char selection[40];
char ch;
//character
int i;
//position in the array
```

APPENDIX D

Saturation Irradiation 1 Data

Time	Counts	Time	Counts	Time	Counts
0.025	20,040	0.875	16,640	1.725	15,520
0.050	21,840	0.900	17,760	1.750	16,280
0.075	21,440	0.925	17,840	1.775	14,880
0.100	19,840	0.950	18,040	1.800	16,240
0.125	21,040	0.975	17,480	1.825	14,240
0.150	20,800	1.000	17,040	1.850	14,960
0.175	21,040	1.025	15,560	1.875	15,760
0.200	22,400	1.050	16,560	1.900	14,000
0.225	20,920	1.075	17,320	1.925	16,080
0.250	19,640	1.100	14,760	1.950	15,040
0.275	20,360	1.125	16,800	1.975	14,200
0.300	20,040	1.150	16,760	2.000	14,600
0.325	19,000	1.175	17,080	2.025	13,280
0.350	20,560	1.200	15,440	2.050	14,520
0.375	19,040	1.225	18,120	2.075	14,600
0.400	19,120	1.250	15,440	2.100	15,360
0.425	20,240	1.275	15,000	2.125	13,640
0.450	18,440	1.300	15,600	2.150	13,200
0.475	18,200	1.325	16,880	2.175	14,280
0.500	18,640	1.350	16,200	2.200	14,600
0.525	17,080	1.375	17,080	2.225	13,800
0.550	18,640	1.400	15,480	2.250	14,400
0.575	18,040	1.425	16,240	2.275	12,720
0.600	18,440	1.450	17,000	2.300	13,520
0.625	19,360	1.475	15,720	2.325	15,360
0.650	18,360	1.500	15,120	2.350	13,560
0.675	19,240	1.525	15,520	2.375	13,920
0.700	17,160	1.550	16,640	2.400	13,800
0.725	18,200	1.575	15,720	2.425	14,320
0.750	18,000	1.600	15,600	2.450	14,240
0.775	18,600	1.625	15,720	2.475	13,960
0.800	18,600	1.650	16,200	2.500	13,600
0.825	17,880	1.675	13,960	2.525	13,280
0.850	18,240	1.700	15,120	2.550	13,560

Time	Counts	Time	Counts	Time	Counts
2.575	14,360	3.550	11,960	4.525	10,487
2.600	14,240	3.575	12,120	4.550	10,411
2.625	13,680	3.600	12,280	4.575	10,348
2.650	14,160	3.625	12,280	4.600	9,878
2.675	12,600	3.650	12,720	4.625	10,361
2.700	13,680	3.675	11,920	4.650	10,573
2.725	12,600	3.700	12,000	4.675	10,488
2.750	12,400	3.725	10,600	4.700	10,271
2.775	12,960	3.750	10,680	4.725	10,524
2.800	13,040	3.775	9,640	4.750	10,284
2.825	12,400	3.800	11,720	4.775	9,967
2.850	12,600	3.825	10,880	4.800	10,150
2.875	12,920	3.850	11,440	4.825	10,130
2.900	12,840	3.875	10,640	4.850	10,120
2.925	12,520	3.900	10,880	4.875	10,367
2.950	12,920	3.925	11,440	4.900	10,217
2.975	14,040	3.950	11,880	4.925	10,138
3.000	11,560	3.975	11,040	4.950	10,186
3.025	13,760	4.000	10,709	4.975	9,561
3.050	13,320	4.025	10,933	5.000	9,939
3.075	13,200	4.050	11,151	5.025	10,033
3.100	11,120	4.075	10,303	5.050	9,702
3.125	12,040	4.100	11,151	5.075	10,291
3.150	12,120	4.125	10,378	5.100	9,677
3.175	12,200	4.150	11,125	5.125	9,798
3.200	12,640	4.175	10,955	5.150	9,707
3.225	14,000	4.200	10,542	5.175	10,106
3.250	12,520	4.225	10,361	5.200	9,425
3.275	11,360	4.250	10,578	5.225	9,417
3.300	12,080	4.275	10,188	5.250	9,767
3.325	12,280	4.300	11,155	5.275	9,658
3.350	12,160	4.325	10,269	5.300	9,793
3.375	12,800	4.350	10,527	5.325	9,659
3.400	11,360	4.375	10,660	5.350	9,424
3.425	12,680	4.400	10,630	5.375	9,971
3.450	10,160	4.425	10,582	5.400	9,307
3.475	11,800	4.450	10,698	5.425	9,503
3.500	11,480	4.475	10,503	5.450	9,811
3.525	11,120	4.500	10,556	5.475	9,413

Time	Counts	Time	Counts	Time	Counts
5.500	9,418	6.475	8,537	7.450	7,980
5.525	9,156	6.500	8,853	7.475	7,638
5.550	9,374	6.525	8,784	7.500	8,507
5.575	9,456	6.550	8,722	7.525	7,609
5.600	9,139	6.575	8,058	7.550	7,732
5.625	9,485	6.600	8,394	7.575	8,052
5.650	9,504	6.625	8,511	7.600	7,862
5.675	9,576	6.650	8,292	7.625	7,522
5.700	9,101	6.675	8,306	7.650	8,177
5.725	9,258	6.700	8,403	7.675	7,856
5.750	9,069	6.725	8,354	7.700	7,925
5.775	8,833	6.750	8,274	7.725	7,852
5.800	9,163	6.775	9,086	7.750	7,759
5.825	9,043	6.800	8,842	7.775	7,648
5.850	9,160	6.825	8,174	7.800	7,904
5.875	8,872	6.850	8,582	7.825	7,897
5.900	9,397	6.875	8,415	7.850	7,888
5.925	9,227	6.900	7,537	7.875	7,596
5.950	9,231	6.925	8,325	7.900	7,798
5.975	9,109	6.950	8,587	7.925	7,691
6.000	9,168	6.975	8,224	7.950	7,720
6.025	9,290	7.000	8,767	7.975	7,503
6.050	9,139	7.025	8,252	8.000	7,853
6.075	9,084	7.050	8,427	8.025	7,836
6.100	9,150	7.075	7,820	8.050	7,438
6.125	8,657	7.100	8,521	8.075	7,354
6.150	9,038	7.125	8,369	8.100	7,590
6.175	8,804	7.150	8,026	8.125	7,741
6.200	8,874	7.175	8,117	8.150	7,807
6.225	8,715	7.200	7,988	8.175	7,723
6.250	8,957	7.225	8,499	8.200	7,672
6.275	8,779	7.250	8,090	8.225	7,316
6.300	8,982	7.275	8,016	8.250	7,799
6.325	8,369	7.300	7,940	8.275	7,442
6.350	8,630	7.325	8,054	8.300	7,202
6.375	8,394	7.350	7,599	8.325	7,178
6.400	8,662	7.375	8,055	8.350	7,633
6.425	8,612	7.400	8,077	8.375	7,455
6.450	8,642	7.425	8,149	8.400	7,479

Time	Counts	Time	Counts	Time	Counts
8.425	7,436	9.400	6,725	17.500	4,617
8.450	7,469	9.425	6,704	18.000	4,563
8.475	7,257	9.450	6,354	18.500	4,443
8.500	7,729	9.475	6,434	19.000	4,307
8.525	7,667	9.500	7,017	19.500	4,292
8.550	7,025	9.525	6,596	20.000	4,113
8.575	7,010	9.550	7,196	20.500	4,098
8.600	7,168	9.575	7,149	21.000	3,999
8.625	7,212	9.600	6,790	21.500	3,949
8.650	7,247	9.625	6,822	22.000	3,842
8.675	7,425	9.650	6,901	22.500	3,787
8.700	7,020	9.675	6,789	23.000	3,687
8.725	7,205	9.700	6,996	23.500	3,680
8.750	7,136	9.725	7,058	24.000	3,601
8.775	7,651	9.750	6,702	24.500	3,568
8.800	6,735	9.775	6,691	25.000	3,466
8.825	7,338	9.800	7,015	25.500	3,405
8.850	6,609	9.825	6,759	26.000	3,374
8.875	7,443	9.850	6,500	26.500	3,289
8.900	7,199	9.875	6,551	27.000	3,173
8.925	6,898	9.900	6,832	27.500	3,155
8.950	7,110	9.925	6,930	28.000	3,057
8.975	6,918	9.950	6,611	28.500	3,039
9.000	7,499	9.975	6,536	29.000	3,104
9.025	6,968	10.000	6,361	29.500	2,963
9.050	6,810	10.500	6,599	30.000	2,920
9.075	7,174	11.000	6,415	30.500	2,843
9.100	7,053	11.500	6,116	31.000	2,836
9.125	6,772	12.000	5,998	31.500	2,853
9.150	7,030	12.500	5,798	32.000	2,796
9.175	7,120	13.000	5,647	32.500	2,743
9.200	7,182	13.500	5,524	33.000	2,642
9.225	6,609	14.000	5,350	33.500	2,649
9.250	6,838	14.500	5,243	34.000	2,659
9.275	6,730	15.000	5,156	34.500	2,569
9.300	6,845	15.500	4,999	35.000	2,499
9.325	6,704	16.000	4,959	35.500	2,471
9.350	6,842	16.500	4,705	36.000	2,402
9.375	7,178	17.000	4,792	36.500	2,414

Time	Counts	Time	Counts	Time	Counts
37.000	2,305	56.500	1,351	76.000	889
37.500	2,371	57.000	1,395	76.500	823
38.000	2,266	57.500	1,340	77.000	818
38.500	2,199	58.000	1,314	77.500	840
39.000	2,233	58.500	1,321	78.000	803
39.500	2,243	59.000	1,312	78.500	811
40.000	2,112	59.500	1,315	79.000	800
40.500	2,170	60.000	1,262	79.500	786
41.000	2,129	60.500	1,249	80.000	762
41.500	2,029	61.000	1,247	80.500	741
42.000	2,065	61.500	1,196	81.000	782
42.500	2,052	62.000	1,168	81.500	747
43.000	2,000	62.500	1,153	82.000	718
43.500	1,993	63.000	1,171	82.500	728
44.000	1,907	63.500	1,165	83.000	723
44.500	1,901	64.000	1,115	83.500	737
45.000	1,899	64.500	1,130	84.000	693
45.500	1,863	65.000	1,111	84.500	677
46.000	1,799	65.500	1,124	85.000	676
46.500	1,807	66.000	1,092	85.500	670
47.000	1,761	66.500	1,069	86.000	661
47.500	1,751	67.000	1,047	86.500	635
48.000	1,707	67.500	1,048	87.000	641
48.500	1,707	68.000	1,040	87.500	621
49.000	1,694	68.500	1,005	88.000	629
49.500	1,658	69.000	990	88.500	627
50.000	1,638	69.500	992	89.000	625
50.500	1,604	70.000	1,008	89.500	627
51.000	1,606	70.500	970	90.000	623
51.500	1,598	71.000	939	90.500	589
52.000	1,557	71.500	943	91.000	623
52.500	1,532	72.000	939	91.500	587
53.000	1,507	72.500	930	92.000	554
53.500	1,487	73.000	900	92.500	574
54.000	1,465	73.500	862	93.000	563
54.500	1,410	74.000	874	93.500	567
55.000	1,453	74.500	857	94.000	555
55.500	1,418	75.000	858	94.500	537
56.000	1,403	75.500	836	95.000	546

Time	Counts	Time	Counts	Time	Counts
95.500	562	155.000	175	255.000	39
96.000	522	160.000	158	260.000	34
96.500	529	165.000	142	265.000	34
97.000	518	170.000	136	270.000	30
97.500	506	175.000	122	275.000	28
98.000	521	180.000	113	280.000	27
98.500	496	185.000	104	285.000	25
99.000	498	190.000	97	290.000	23
99.500	468	195.000	86	295.000	22
100.000	477	200.000	84	300.000	20
105.000	452	205.000	75	305.000	20
110.000	413	210.000	69	310.000	19
115.000	370	215.000	65	315.000	17
120.000	338	220.000	60	320.000	16
125.000	303	225.000	56	325.000	16
130.000	276	230.000	54	330.000	14
135.000	247	235.000	49	335.000	13
140.000	224	240.000	48	340.000	13
145.000	203	245.000	41	345.000	12
150.000	188	250.000	40	350.000	11

APPENDIX E

Burst Irradiation 1 Data

Time	Counts	Time	Counts	Time	Counts
0.025	53,864	0.875	30,360	1.725	22,880
0.050	43,320	0.900	30,440	1.750	23,760
0.075	43,680	0.925	29,680	1.775	23,280
0.100	44,600	0.950	30,400	1.800	21,720
0.125	44,280	0.975	29,960	1.825	22,400
0.150	44,120	1.000	31,320	1.850	21,480
0.175	43,000	1.025	30,840	1.875	22,200
0.200	42,440	1.050	29,920	1.900	23,440
0.225	42,560	1.075	27,960	1.925	22,080
0.250	41,240	1.100	29,080	1.950	22,280
0.275	41,640	1.125	26,960	1.975	22,800
0.300	39,120	1.150	28,000	2.000	20,680
0.325	38,600	1.175	27,280	2.025	21,320
0.350	39,080	1.200	29,320	2.050	20,520
0.375	38,280	1.225	27,200	2.075	20,360
0.400	40,040	1.250	27,640	2.100	19,720
0.425	38,640	1.275	29,040	2.125	20,520
0.450	36,600	1.300	25,200	2.150	21,920
0.475	35,960	1.325	27,240	2.175	21,040
0.500	34,880	1.350	25,160	2.200	20,440
0.525	34,600	1.375	25,360	2.225	20,200
0.550	34,160	1.400	24,560	2.250	20,680
0.575	34,640	1.425	26,640	2.275	19,240
0.600	32,520	1.450	24,640	2.300	18,720
0.625	33,320	1.475	25,640	2.325	19,240
0.650	35,080	1.500	25,160	2.350	18,800
0.675	34,200	1.525	24,400	2.375	19,240
0.700	31,360	1.550	25,040	2.400	20,160
0.725	31,160	1.575	23,600	2.425	20,120
0.750	31,880	1.600	25,120	2.450	18,640
0.775	31,320	1.625	22,960	2.475	17,680
0.800	32,600	1.650	23,240	2.500	19,920
0.825	31,320	1.675	23,760	2.525	18,960
0.850	30,640	1.700	23,360	2.550	18,160

Time	Counts	Time	Counts	Time	Counts
2.575	17,400	3.550	14,760	4.525	10,880
2.600	18,440	3.575	13,880	4.550	12,680
2.625	18,480	3.600	15,320	4.575	10,960
2.650	17,920	3.625	14,240	4.600	11,880
2.675	17,680	3.650	13,840	4.625	10,720
2.700	18,000	3.675	14,000	4.650	11,400
2.725	17,440	3.700	13,880	4.675	12,080
2.750	18,080	3.725	13,840	4.700	10,720
2.775	17,320	3.750	12,760	4.725	10,800
2.800	17,720	3.775	14,880	4.750	11,720
2.825	17,720	3.800	13,160	4.775	11,480
2.850	16,720	3.825	14,000	4.800	11,200
2.875	17,960	3.850	13,640	4.825	11,400
2.900	17,560	3.875	13,200	4.850	11,080
2.925	15,440	3.900	14,480	4.875	10,480
2.950	16,280	3.925	12,960	4.900	10,440
2.975	17,640	3.950	13,280	4.925	10,440
3.000	16,640	3.975	12,040	4.950	10,000
3.025	15,680	4.000	13,280	4.975	10,400
3.050	16,000	4.025	14,080	5.000	10,720
3.075	16,640	4.050	13,600	5.025	10,680
3.100	16,440	4.075	12,000	5.050	10,680
3.125	16,040	4.100	12,800	5.075	11,280
3.150	15,920	4.125	12,600	5.100	10,640
3.175	16,440	4.150	12,280	5.125	10,120
3.200	16,800	4.175	12,600	5.150	9,840
3.225	15,400	4.200	11,360	5.175	9,800
3.250	15,200	4.225	13,160	5.200	10,240
3.275	13,760	4.250	12,920	5.225	10,280
3.300	14,920	4.275	11,920	5.250	9,440
3.325	14,960	4.300	13,120	5.275	10,640
3.350	15,400	4.325	12,000	5.300	9,920
3.375	15,680	4.350	12,560	5.325	9,760
3.400	15,880	4.375	12,720	5.350	10,440
3.425	14,800	4.400	13,080	5.375	9,680
3.450	14,280	4.425	11,560	5.400	9,640
3.475	15,720	4.450	11,440	5.425	10,600
3.500	13,880	4.475	12,640	5.450	9,040
3.525	14,160	4.500	11,400	5.475	10,200

Time	Counts	Time	Counts	Time	Counts
5.500	9,680	6.475	8,320	7.450	6,606
5.525	8,800	6.500	8,000	7.475	6,712
5.550	9,360	6.525	8,080	7.500	6,550
5.575	9,200	6.550	8,280	7.525	6,283
5.600	8,960	6.575	8,640	7.550	6,402
5.625	9,440	6.600	8,600	7.575	5,982
5.650	9,040	6.625	7,560	7.600	6,561
5.675	9,320	6.650	7,160	7.625	6,269
5.700	9,120	6.675	7,960	7.650	6,155
5.725	9,880	6.700	7,640	7.675	6,482
5.750	9,240	6.725	7,040	7.700	6,337
5.775	9,000	6.750	8,240	7.725	6,519
5.800	8,000	6.775	7,240	7.750	6,419
5.825	9,240	6.800	7,280	7.775	6,381
5.850	8,160	6.825	8,200	7.800	6,610
5.875	9,160	6.850	7,120	7.825	6,397
5.900	8,600	6.875	7,160	7.850	6,523
5.925	8,720	6.900	7,480	7.875	6,288
5.950	9,000	6.925	7,120	7.900	6,246
5.975	8,560	6.950	7,520	7.925	5,907
6.000	8,680	6.975	6,520	7.950	5,992
6.025	8,120	7.000	7,283	7.975	6,024
6.050	9,240	7.025	7,537	8.000	6,250
6.075	8,240	7.050	6,968	8.025	6,021
6.100	8,360	7.075	7,571	8.050	6,160
6.125	8,840	7.100	6,809	8.075	6,011
6.150	9,160	7.125	6,435	8.100	5,833
6.175	8,640	7.150	7,055	8.125	5,819
6.200	8,640	7.175	7,014	8.150	5,797
6.225	8,240	7.200	7,008	8.175	6,196
6.250	9,040	7.225	7,016	8.200	5,731
6.275	8,240	7.250	6,977	8.225	5,765
6.300	7,960	7.275	6,621	8.250	5,773
6.325	8,440	7.300	6,741	8.275	5,547
6.350	8,760	7.325	6,801	8.300	5,816
6.375	9,080	7.350	6,822	8.325	5,734
6.400	8,280	7.375	6,603	8.350	5,225
6.425	7,920	7.400	6,583	8.375	5,644
6.450	8,920	7.425	6,552	8.400	6,020

Time	Counts	Time	Counts	Time	Counts
8.425	5,736	9.400	5,218	17.500	2,132
8.450	5,431	9.425	5,034	18.000	2,088
8.475	5,311	9.450	4,930	18.500	2,028
8.500	5,346	9.475	4,572	19.000	1,873
8.525	5,613	9.500	4,640	19.500	1,846
8.550	5,446	9.525	5,177	20.000	1,772
8.575	5,360	9.550	4,933	20.500	1,712
8.600	5,298	9.575	4,643	21.000	1,707
8.625	5,843	9.600	4,485	21.500	1,627
8.650	5,656	9.625	4,887	22.000	1,624
8.675	5,620	9.650	4,906	22.500	1,536
8.700	5,261	9.675	4,776	23.000	1,499
8.725	5,338	9.700	5,051	23.500	1,475
8.750	5,229	9.725	4,895	24.000	1,457
8.775	5,217	9.750	4,640	24.500	1,394
8.800	5,188	9.775	4,649	25.000	1,325
8.825	5,366	9.800	4,654	25.500	1,357
8.850	5,378	9.825	4,786	26.000	1,234
8.875	5,289	9.850	4,592	26.500	1,238
8.900	5,032	9.875	4,566	27.000	1,213
8.925	5,360	9.900	4,440	27.500	1,193
8.950	5,277	9.925	4,299	28.000	1,164
8.975	5,221	9.950	4,721	28.500	1,146
9.000	5,173	9.975	4,577	29.000	1,101
9.025	5,411	10.000	4,509	29.500	1,066
9.050	5,276	10.500	4,411	30.000	1,020
9.075	5,208	11.000	4,096	30.500	1,002
9.100	4,984	11.500	3,863	31.000	1,005
9.125	4,948	12.000	3,655	31.500	977
9.150	4,981	12.500	3,434	32.000	953
9.175	5,039	13.000	3,210	32.500	939
9.200	5,345	13.500	3,041	33.000	897
9.225	5,269	14.000	3,004	33.500	887
9.250	4,935	14.500	2,830	34.000	866
9.275	5,068	15.000	2,731	34.500	880
9.300	5,165	15.500	2,563	35.000	839
9.325	4,922	16.000	2,466	35.500	842
9.350	5,117	16.500	2,417	36.000	820
9.375	5,011	17.000	2,299	36.500	800

Time	Counts	Time	Counts	Time	Counts
37.000	803	56.500	425	76.000	226
37.500	755	57.000	415	76.500	234
38.000	738	57.500	389	77.000	224
38.500	745	58.000	420	77.500	219
39.000	701	58.500	383	78.000	200
39.500	670	59.000	383	78.500	220
40.000	701	59.500	368	79.000	269
40.500	688	60.000	358	79.500	200
41.000	694	60.500	365	80.000	199
41.500	681	61.000	344	80.500	204
42.000	659	61.500	347	81.000	209
42.500	666	62.000	341	81.500	207
43.000	659	62.500	360	82.000	185
43.500	627	63.000	353	82.500	187
44.000	630	63.500	323	83.000	192
44.500	585	64.000	354	83.500	176
45.000	618	64.500	328	84.000	187
45.500	565	65.000	306	84.500	179
46.000	576	65.500	304	85.000	179
46.500	573	66.000	311	85.500	186
47.000	561	66.500	293	86.000	180
47.500	531	67.000	300	86.500	177
48.000	530	67.500	314	87.000	169
48.500	508	68.000	279	87.500	171
49.000	535	68.500	298	88.000	157
49.500	536	69.000	281	88.500	159
50.000	517	69.500	271	89.000	168
50.500	495	70.000	280	89.500	156
51.000	509	70.500	269	90.000	165
51.500	503	71.000	268	90.500	149
52.000	452	71.500	257	91.000	165
52.500	458	72.000	250	91.500	166
53.000	454	72.500	247	92.000	153
53.500	468	73.000	258	92.500	144
54.000	444	73.500	260	93.000	145
54.500	409	74.000	249	93.500	148
55.000	448	74.500	224	94.000	135
55.500	418	75.000	236	94.500	134
56.000	409	75.500	233	95.000	139

Time	Counts	Time	Counts	Time	Counts
95.500	144	155.000	37	255.000	7
96.000	139	160.000	33	260.000	5
96.500	120	165.000	28	265.000	5
97.000	135	170.000	26	270.000	5
97.500	123	175.000	22	275.000	5
98.000	131	180.000	21	280.000	5
98.500	126	185.000	20	285.000	4
99.000	130	190.000	17	290.000	4
99.500	111	195.000	16	295.000	3
100.000	110	200.000	16	300.000	3
105.000	112	205.000	13	305.000	3
110.000	100	210.000	12	310.000	3
115.000	85	215.000	12	315.000	3
120.000	77	220.000	10	320.000	2
125.000	66	225.000	10	325.000	2
130.000	62	230.000	9	330.000	2
135.000	54	235.000	8	335.000	2
140.000	48	240.000	7	340.000	2
145.000	44	245.000	7	345.000	2
150.000	38	250.000	6	350.000	2

VITA

Aaron David Heinrich honorably served in the United States Navy from May 1998 to May 2004. During that time he graduated from the Electrician's Mate Nuclear Field 'A' School and Naval Nuclear Power School in Charleston, South Carolina, and the Nuclear Propulsion Training Unit in Ballston Spa, New York. He was stationed onboard an Ohio-class ballistic missile submarine in Kings Bay, Georgia, and performed seven strategic deterrence missions.

He received his Bachelor of Science in Applied Science and Technology with an emphasis in Nuclear Engineering Technology from Thomas Edison State College in September 2005, and entered the Nuclear Engineering program at Texas A&M University in August 2005. In May 2008 he received his Master of Science degree.

Mr. Heinrich may be reached at the South Texas Project Nuclear Operating Company, STP Units 3 and 4, 4000 Avenue F, Bay City, TX 77414-7612. His email address is ADHeinrich@STPEGS.com.